"SOFTWARE TOOLS FOR EXPERIMENTING WITH CELLULAR AUTOMATA"

by

INWHAN CHOI

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF
BACHELOR OF SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER, 1982

Signature of Author .................................................. Signature redacted

Department of Electrical Engineering and Computer Science

Certified by ............................................................... Signature redacted

Thesis Supervisor

Accepted by .............................................................. Signature redacted

Chairman, Department Committee on Thesis

Archives

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

NOV 1 1982

LIBRARIES
ABSTRACT

A program to control a hardware cellular automaton simulator was written in 6502 Assembly Language and BASIC on an ATARI microcomputer. The program allows the user to set up initial configurations for the simulator, download transition rules and apply the downloaded transition rule by sending the appropriate commands to the simulator from the microcomputer. Further, it is possible to save interesting configurations through the use of the program. An evaluation of the utility of the control program in experimenting with cellular automata is made.

Thesis Supervisor: Edward Fredkin, Professor of Electrical Engineering.
ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Mr. Norman Margolus who was most generous with his time in explaining his control program on which the author's work is based. Without his proffered knowledge of the Atari microcomputer this project would not have been completed. I would also like to thank Dr. Tommaso Toffoli for explaining to me how the simulator works.
CONTENTS

1. Introduction ........................................................................... 5

2. Simulator Control Program Description ................................. 8
   2.1 Atari-Simulator Hardware Interface ................................. 8
   2.2 Assembly Language Routines ....................................... 11
   2.3 BASIC Control Program ............................................. 17

3. Conclusions .......................................................................... 21

4. References ........................................................................... 22

Appendix I. Program ................................................................. 23
1. Introduction

A cellular automaton is a space distributed dynamical system whose evolution law is uniform and local. It is a stylization of a physical universe. In this stylization, the "universe" consists of a uniform checkerboard, with each square or cell being in one of finitely many states; time advances in discrete steps and the "laws of the universe" are a small look-up table through which each cell on the checkerboard calculates its next state from the state of its neighbors. That is, the next state of the board is obtained by applying a local transition rule to each cell on the board simultaneously.

Cellular automata were invented by John von Neumann in order to provide a general framework for proving the possibility of self-replicating automata. Von Neumann, found a cellular automaton where each cell had one of 29 possible states, with a transition rule using the four orthogonal neighbors, that could replicate itself.¹

The importance of cellular automata, however, lies in their connection with computation and physics. It is well known that very simple cellular automata exist which are computation universal, that is, one can simulate any process within the framework of these cellular automata.² Furthermore, it has recently been shown that the functional behavior of a general-purpose computer can be reproduced by an ideal gas placed in a suitably shaped container and given

¹. [Gardener 71]
². Banks Rule on five neighbors, listed in the BASIC program, is an example.
appropriate initial conditions. In this Billiard Ball Model (BBM) model of computation, concepts such as energy and entropy have a direct computational meaning. This result is significant with respect to cellular automata since it turns out that the Billiard Ball Model of computation is isomorphic to a cellular automaton. In fact, the cellular automaton that is isomorphic to the Billiard Ball Model of Computation is reversible, by which we mean that the present state uniquely determines the past and future state. This is very similar to microscopic laws in physics where knowing the initial conditions and the governing laws allows one to calculate the time evolution of that system in both directions. Thus, it seems that the study of the workings of cellular automata is similar to the study of physics.

Simulating cellular automata by using a general purpose computer is too slow. A more dedicated and faster cellular automaton simulator is needed, to investigate the above ideas by direct experimentation. Such a cellular automaton simulator, implemented in hardware, was designed by Dr. Tommaso Toffoli at the M.I.T. Laboratory for Computer Science. With this hardware simulator one can set up an arbitrary initial configuration, an arbitrary transition rule, and display the result on a CRT monitor in real time (60 frames per second). The simulator was designed to provide an easy interface to the Motorola 6520/6820 Parallel Interface Adaptor chip.

The aim of this project was to write a controlling program on the Atari

1. [Fredkin 81]
microcomputer (which has a 6820 at its disposal) so as to make it easier to use the simulator to investigate cellular automata. The program makes the hardware details of the simulator transparent to the user.
2. Simulator Control Program Description

2.1 Atari-Simulator Hardware Interface

In this simulator, the size of the "universe" is 256 cells by 256 cells, stored in a "memory plane" of 8K bytes. Here, we shall discuss the use of only two such memory planes. One memory plane holds the current configuration of the cellular automaton and the other holds the previous configuration of the cellular automaton. The two planes are bit-mapped on to the CRT monitor to yield one of four grey levels at each pixel of the display which has 256 usable raster-scan lines with 256 pixels per line. Thus, four states per cell are possible with the use of two memory planes. The simulator provides for up to eight memory planes so that up to 256 states per cell is possible. The hardware of the simulator is configured to support eight different transition tables but we shall discuss the use of only one transition table. The control program assumes that all eight memory planes and all eight transition tables exist so that no changes are needed in the software when the simulator is upgraded to have more memory planes and transition tables. It was found to be useful to have a buffer in the Atari that acts as an image for the memory planes that hold the configuration of the cellular automaton. This allows the contents of a memory plane, a slice of the configuration, to be more easily read or written to by the Atari.

The simulator has four main modes of operation. They are as follows.

Mode 1: Read memory plane to Atari buffer
Mode 2: Write Atari buffer to memory plane

Mode 3: Download a transition table from the Atari

Mode 4: Perform one of the eight functions of the simulator

Four of the eight functions of the simulator shift the configuration of the cellular automaton to the left, right, down and up. The other four functions can be derived from either the contents of a RAM table or by combinational circuits. Currently, one function places a box in the middle of the simulator display, another function runs John Conway's "Life", a third function applies the transition rule that is currently downloaded and the last function applies the current downloaded rule and then exor's with the previous state to get the next state of a cell.

Communication to and from the Atari microcomputer to the cellular automaton simulator interface occurs through Port A and Port B of the 6820 PIA chip. Also, the CB2 peripheral control line of the 6820 is utilized to generate a synchronization signal to the simulator. There are six registers on the 6820 that are accessible to the MPU: two peripheral data registers, Port A and Port B, two data direction registers (one for each port) and two control registers (one for each port). Each of the 8 lines on both Port A and Port B can be configured for input or output corresponding to what bits are set in their respective data direction registers. The two control registers PACTL and PBCTL also allow the MPU to control the operation of the four peripheral control lines CA1, CA2, CB1, and CB2.
There are four types of commands that can be sent to the hardware simulator, corresponding to the four types of operation described previously. One command type is to read the memory planes of the simulator. The inverse command to write to the memory plane can also be sent. By being able to read or write to the memory planes of the simulator from the Atari, we can save a configuration of the cellular automaton on disk or download an arbitrary configuration. Another command is to send to the simulator the transition table to calculate the next state of each cell depending on the state of its four orthogonal neighbors and itself. Both the hardware of the simulator and controlling software can be quickly changed to send the simulator a transition table that computes the next state of each cell depending on the state of its nine nearest neighbors including itself, however. Finally there is a command that tells the simulator to perform one of eight functions that the simulator supports. These eight functions have been previously described.

All four commands are sent to the simulator from the Atari in exactly the same way. By convention, commands are sent through Port A and data through Port B. Bit A6 (MSB-1) of Port A is used as a strobe to tell the simulator that a command is being sent. On the rising edge of A6, the low order 6 bits (A5-A0) of Port A, which hold the command and its parameters are latched by the simulator. The simulator decodes which command is being sent by decoding bits A5 and A4. The low order four bits specify which memory plane is being addressed, or the transition table being written to, or one of the eight hardware functions being requested of the simulator. When the simulator has finished processing the command it sends an acknowledge signal by setting its line to A7 of Port A, which
has been configured for input, high. Each of the commands except requests to perform one of the eight simulator functions is accompanied by a data transfer. Data transfers occur through Port B and are controlled by the CB2 peripheral control line. By writing the control "word" 2C hexadecimal to the Port B control register, the CB2 line of the 6820 goes low on the first CLK pulse following an MPU write to the Port B data register. CB2 then goes high on the next CLK pulse to the 6820. Thus a strobe is generated every time data is written to Port B. Depending on the current command, data is transferred to the various subsystems of the simulator from the Atari. During an output operation the appropriate subsystem utilizes the CB2 strobe to latch the data which is then available on Port B. During an input operation the Atari reads the data which is placed on the Port B lines by the simulator and writes this back to Port B. Since Port B is configured for input at this stage it does not change the data at the simulator. However, the strobe on CB2 is still produced because an MPU write to Port B has taken place. The simulator then uses the strobe to place the next byte of data on the Port B lines. Communication between the Atari and the simulator may be disabled by setting bit A6 of Port A high.

2.2 Assembly Language Routines

The control program is divided into two parts, one part is written in 6502 Assembly Language and the other part is written in BASIC. The assembly language routines perform the actual I/O between the Atari microcomputer and the simulator. The BASIC language routines essentially provide the user interface
and controls the simulator by calling the appropriate assembly language routines. The object code for the assembly language routines is stored in a file on disk called AUTORUN.SYS. The disk operating system of the Atari will load the contents of this file, if it exists, on the disk into the memory of the Atari every time the system is booted. Another aspect of the AUTORUN.SYS facility on the Atari is that the programmer can specify that certain segments of the code should be executed as soon as it is loaded into memory. The other segments are executed for the first time when they are called by some routine. This facility has been used with the assembly language portion of the simulator control program (and is more fully documented in the assembly language listing) to perform initialization. First, the system initialization vector is changed to point to a new initialization routine. This new routine initializes the simulator control program and then performs the normal system initialization. Thus, when a <Return> command is executed the simulator control program is initialised along with the Atari.

The initialization part of the assembly language routines reserves all memory above 7000 hexadecimal for the display buffer and the assembly language routines. The display buffer is 8K locations, (each location holds a byte), in length and is used to read the configuration of the cellular automata or to write a new configuration to the simulator memory plane. Further, we create a display list in a high part of memory for future use. A display list is a set of instructions to the Antic video processor which is a part of the Atari microcomputer, specifying where to find the memory for the Atari screen display and how to interpret the values in the screen RAM. Originally, it was intended to be used to overlay an enlargement of a section of the simulator display on the
terminal screen but has not been implemented for a variety of reasons. Currently, it serves no useful purpose. Finally, the background control process for the hardware simulator is initialized. This is one of the most crucial parts of the simulator control program. The background control process is called as a timed-interrupt routine. The system vector for the countdown timer CDTMV2 is changed to point to the routine for the background process. Every 1/60 of a second the timer CDTMV2 is decremented. If after decrementing, the count reaches zero, then the background process is called. Then the process sends a command to the simulator to perform one of its eight hard-wired functions. The other three types of commands on the simulator are useful mainly for setting up initial configurations and downloading transition rules. One can’t actually make the simulator "do something" with these commands. However, once the initial configuration is set and the transition rule is downloaded one would like the simulator to apply the transition table to the configuration, or move it to the right and so on. And one would like the simulator to continue to perform each of these commands until one wishes to stop that command. Two variables, NUMFNC and PERIOD, are passed as arguments to the background control process with BASIC poke statements. The background process uses NUMFNC to decide which function to request of the simulator. PERIOD determines the frequency with which the background process is called. Before the call to the background process terminates it loads the countdown timer CDTMV2 with the contents of PERIOD. Hence, by poking the appropriate value into PERIOD, one can stop the simulator or change the speed at which the simulator is running.

The following is a descriptions of the other assembly language routines
that have a hand in controlling the simulator.

The subroutine SETDAT sets up data transfer from the Atari to the simulator. It assumes that it is being called from BASIC and hence it expects that both the command and the direction of data transfer is passed to it as arguments on the stack. The routine also has an assembly language entry point where the parameters to the routine are passed to it in the 6502 registers. Many of the assembly language routines have this feature. The command is sent to the simulator by calling the subroutine COMMND. The direction for data transfer is then set by writing the direction value to the Port B data direction control register. Finally, the control "word" 2C hexadecimal is written to the Port B control register to generate a strobe on the CB2 peripheral control line as explained previously.

The routine COMMND assumes that the command to be sent to the simulator is passed to it in the lower six bits of the accumulator. COMMND initialises the 6820 and sends the command through Port A by toggling bit A6 of Port A to strobe the command in the lower 6 bits of Port A to the simulator.

The routine STPDAT disables data transfer from the Atari to the simulator by preventing the toggling of bit A6 by setting it high and by setting the direction control registers for input.

The routines WRITED and RDDSP perform inverse functions with respect to each other. The routine WRITED writes the display buffer in the Atari memory to the memory planes of the simulator. Hence, one can place any desired configuration on the simulator display by placing the configuration in the display buffer and calling the routine WRITED. The routine RDDSP reads the
configuration from a memory plane of the simulator and stores it in the display buffer.

The contents of the display buffer, the configuration, can be saved on disk by calling the routine SAVBUF after an "OPEN.." command in BASIC. The routine SAVBUF uses the Centralized Input/Output (CIO) facility of the Atari's operating system.

The CIO facility provides for device-independent data transfer. From the point of view of the operating system, I/O is organised around a standard table, called an Input Output Control Block (IOCB), that completely specifies some input or output operation. A partial list of an IOCB appears on the first page of the assembly listing. There are eight IOCB's starting at 0340 hexadecimal. Each IOCB is 16 locations in length. In most cases, the user sets up an IOCB with the appropriate control data in the group of 16 locations chosen and then passes control to the CIO facility which does the rest.

For example, the basic command: OPEN #3,3,0,"D:BUF" would associate IOCB #3 (starting at 370 hexadecimal) with the file in the disk called "D:BUF". The value of the second argument tells the CIO facility that it is an output operation to that file. The second argument of the OPEN command would be 4 for input. Now, by changing the control data in IOCB #3 we can perform various I/O operations (all of an output nature to the file D:BUF). If we set the command byte (which would be located at 372 hexadecimal for IOCB #3) to 0B hexadecimal, the "put characters" command and set the buffer addresses (located at 374 and 375 hexadecimal for IOCB #3) to the start of the display buffer and if the length of the display buffer is placed in the locations for the
buffer length for IOCB #3 (which would be at 378 and 379 hexadecimal) the contents of the display buffer is written a byte at a time to the file D:BUF. The SAVBUF routine expects the number of the IOCB used to be passed to it as an argument so that it can calculate where to place control data changes for that IOCB. The LOADBF routine is identical except that the command byte is 07 hexadecimal, or the "get characters command". Further, the second argument to the OPEN statement is a 4 to specify an input operation. When I/O is finished, a "CLOSE..." statement is needed.

If the above explanation is not sufficiently clear, the reader is encouraged to consult the operating system manual for the Atari microcomputer for further clarification.

Finally, there are two routines that are used in setting up the initial configuration of the simulator. SETPLN writes the value, passed to it on the stack by a BASIC call, to the simulator memory plane specified, which is also placed on the stack as an argument to the call. Unfortunately, at the moment, one can only specify an area that is one pixel high by eight pixels wide (1 byte) using this command. The simulator display will be formed from this "atomic" unit.

The more useful routine is INITSN which AND's, OR's, or EXOR's the display buffer with the memory plane specified, and returns the configuration to the display buffer. By using INITSN judiciously, one can generate a fairly large variety of initial conditions. The routine WRITED is called to send the display buffer to the simulator display once one is satisfied with the contents of the display. A BASIC routine version of INITSN does this automatically for the user.
2.3 BASIC Control Program

The part of the control program that is written in BASIC controls the cellular simulator at a higher functional level. It provides the user interface and calls the appropriate assembly language routines to carry out the user requests concerning the cellular automaton. The initialization section defines variables that hold the addresses of the various assembly language routines. With one exception, (i.e. the use of BUFPLN for INITSN), the variable name refers to the corresponding assembly language routine with the same name. A note should also be made of location 764, KCODE. This location holds the key code for the last character entered. For example, if "A" was the last character entered then location KCODE would hold 63. (The KCODE values were discovered by experimentation).

The user is placed in a variety of command environments by the program, and stays in that environment until a command is received asking to leave that command environment. At the top level of each command environment, user prompts are read in from the disk, giving the user a list of commands that are available in that command environment. Prompts are read in from disk to conserve the limited RAM resources of the microcomputer. This reduces the readability of the code, but by referring to the appropriate prompt file, listed in the appendix, it should not be difficult to follow the program. For example, at the top level of the program the file "D:TOPLVL.PMT" is printed on the Atari screen giving the user very general options. Once the user has chosen an option, control is passed to that command environment and a new list of
options becomes available.

The following is a description of the various command handlers in the BASIC program.

The handler to define and download a new transition rule simply prints on the terminal screen the 10 lines of the program, 30000 to 30010 and exits the program, leaving the user in Atari screen edit mode. The user can then use the screen editor to define the transition rule in those lines of code. This actually changes the RAM-resident program. The user enters "CONT <RETURN>" to resume execution. To download this rule, the download routine is called. It calls the assembly language routine called SETDAT to set up the simulator to accept the transition table.

Then the transition table is generated for this rule. For every possible state of a center cell and its four neighbors, the next state of the center cell is calculated then poked out to Port B. The hardware of the simulator latches the output or Port B only every other time during the downloading of a transition rule and this necessitates another poke Port B command. The transition table that is generated and downloaded, is encoded as a text string and stored in a global variable called SAVE$. Finally, data transfer to the simulator is disabled by a call to the assembly routine STPDAT.

The command handler to save or retrieve transition rules is very simple. To save a definition of the transition rule, the user is asked for a name to save the rule under, and the definition in lines 30000 to 30010 are written to that file. To retrieve the definition of the rule, the user-specified rule (file) is entered from disk, replacing lines 30000 to 30010 of the RAM-resident program with the new rule.
To save the transition table only (which saves space on disk), the string SAVE$ holding the encoded transition table is written to a file with a name specified by the user. To retrieve the table from a user specified file, the content of the file is entered from disk and saved in SAVE$. To download a transition table retrieved in the form of a text string is analogous to the way a transition rule definition was downloaded. Instead of calculating the next state of the center cell and sending it out, the string SAVE$ is decoded one character at a time and sent to the simulator.

The command handler to setup the initial conditions of the simulator merely calls the appropriate assembly language routine, with the exception of the command to place a box in the display buffer. This command is carried out by poking the area of the display buffer that corresponds to the center of the simulator display.

The command handler to run the simulator is also very simple. It pokes the function number of the requested command into NUMFNC and turns on the background process by writing a "1" to the countdown timer, CDTMV2, which enables it. Further, depending on whether the command is to be done once or repeatedly, the location PERIOD is poked with zero, or the period in 1/60 seconds between successive commands being sent. The command to try to reverse the simulator is the only interesting command in the run simulator command environment. If the next state of a center cell is determined, by applying the current transition rule to the the present state of that center cell and exoring it with the previous state of the center cell, this yields a reversible rule as can be seen in the following.
Here, $C(t)$ means the state of a cell at time $t$ and $F[C(t)]$ is the value computed by applying the transition rule to that cell at time $t$. Then:

$$C(t + 1) = F[C(t)] \text{ EXOR } C(t-1)$$

and exoring both sides with $F[C(t)]$ yields:

$$F[C(t)] \text{ EXOR } C(t + 1) = C(t-1)$$

With a change of variable, $T = t + 1$, the equation becomes:

$$F[C(T-1)] \text{ EXOR } C(T) = C(T-2)$$

That is, one can calculate the state before the previous state from the present and previous states. We can reverse the cellular automaton.\(^1\) Thus the command to try to reverse the simulator simply swaps the past state memory plane with the present state memory plane. By running the reversible rule, one should then be able to return to the initial condition of the simulator.

The command handler to read from or write to a memory plane calls the assembly language routine RDDSP and WRITED respectively. The command handler to save the display buffer calls the routine SAVBUF and stores the display buffer in the file specified by the user. To load the buffer from a file specified by the user the routine LOADBF is called.

There is I think sufficient comments for the program to be fairly readable.

---

1. This idea is due to E.Fredkin and N.Margolus.
3. Conclusions

The cellular-automaton control program described here works well enough, particularly, if one is willing to accept a relatively small variety of initial conditions. However, it is a very simple matter to simulate any cellular automaton with a transition rule that is a function of up to 4 adjacent (orthogonal) neighbors and the current state of the center cell by using the program. Further, the program is user-friendly to the extent that one could learn how to use it very quickly. One could increase the variety of initial conditions available for the simulator if one had such functions as writing a random pattern of 1's and 0's to the simulator, or if one could rotate the display for example (and EXOR, AND OR with the display buffer to get a different configuration), and thus improve the program. One could write a subroutine that would make it easy for the user to poke a custom-designed configuration into the display buffer and then write to the simulator. More importantly, I think there is a need for some type of package to do some sort of statistical analysis on the cellular automaton, although it's not clear to me how to go about doing this. There is certainly a lot more work that can be done.

However, I think the main utility of this project lies in its simplicity. I have tried to make it fairly easy for someone to understand the control program described here and to add any commands that the person finds lacking.
4. References


Appendix I - Program
THE FOLLOWING CELLULAR AUTOMATA SIMULATOR CONTROL PROGRAM IS STORED IN A FILE ON DISK CALLED AUTORUN.SYS. THE OPERATING SYSTEM WILL THUS LOAD THIS PROGRAM INTO ATARI RAM EVERY TIME THE SYSTEM IS BOOTTED. THE AUTORUN.SYS FACILITY ALSO ALLOWS THE USER TO SPECIFY THE "INIT" AND "RUN" ADDRESSES OF THE PROGRAM. THE PROGRAM IS EXECUTED BEGINING AT THE "INIT" ADDRESS UNTIL AN RTS INSTRUCTION IS ENCOUNTERED. THEN THE PROGRAM CODE STARTING AT THE "RUN" ADDRESS IS EXECUTED UNTIL A RTS INSTRUCTION IS ENCOUNTERED. ED

THE FOLLOWING SECTION OF THE PROGRAM RESERVES MEMORY, INITIALIZES THE MODE OF THE SCREEN DISPLAY; FURTHER, THE SYSTEM INITIALIZATION VECTOR STORED AT LOCATION DOSINI IS REPLACED WITH OUR OWN INITIALISATION VECTOR. LABELS ARUN AND AINIT ARE THE RUN AND INIT ADDRESSES USED BY THE AUTORUN.SYS FACILITY.

OPERATING SYSTEM EQUATES

```
000C  DOSINI = $000C ;DISK OPERATING SYSTEM INIT ADRS
006A  RAMTOP = $006A ;TOP OF RAM
02E4  RAMSIZE = $02E4 ;SIZE OF RAM
00CB  PGZ = $CB ;FREE PAGE 0 LOC
1430  O.S. ADDRESSES FOR CIO
1440  ICHID = $0340 ;HANDLER ID
1450  ICDNO = ICHID+1 ;DEVICE NUMBER
1460  BOTH OF
1470  ABOVE SET BY CIO
1480  ICCOM = ICDNO+1 ;COMMAND BYTE
1490  ICSTA = ICCOM+1 ;STATUS BYTE SET
1500  BY CIO
1510  ICBAL = ICSTA+1 ;BUFR ADRS (LOW)
1520  ICBALH = ICBAL+1 ;BUFR ADRS (HIGH)
1530  ICPTL = ICBALH+1
1540  ICPTH = ICPTL+1
1550  ICBLL = ICPTH+1 ;BUFFER LENGTH LO
```
0349 1560 ICBLH = ICBLH+1 ; BUFFER LENGTH HI
034A 1570 ICAX1 = ICBLH+1 ; AUX 1
034B 1580 ICAX2 = ICAX1+1 ; AUX 2
1590 ;
0007 1600 GETCH = $07 ; GET CHARS CMMD
000B 1610 PUTCH = $0B ; PUT CHARS CMMD
0003 1620 OPEN = $3 ; IOCB OPEN CMMD
000C 1630 CLOSE = $C ; IOCB CLOSE CMMD
E456 1640 CIOV = $E456 ; CIO ENTRY POINT
1650 ;
E477 1660 COLDSV = $E477 ; COLDBOOT VECTOR
00FF 1670 LO = $00FF ; LOW BYTE MASK
1680 ; ADDRESS&LO=LOW
1690 ; BYTE OF ADDRESS
0100 1700 HI = $0100 ; ADDRESS/HI= HIGH
1710 ; BYTE OF ADDRESS
0228 1720 CDTMA2 = $0228 ; PTR TO TIMER RTN
021A 1730 CDTMV2 = $021A ; COUNTDOWN TIMER
00D4 1740 RESLTL = $D4 ; LOW BYTE OF RSLT
00D5 1750 RESLTH = $D5 ; HI BYTE OF RSLT
00FF 1760 OUT = $FF ; PIA DIR = OUTPUT
0000 1770 IN = $00 ; PIA DIR = INPUT
7000 1780 DBUF = $7000 ; START OF DISP BFR
9C20 1790 DSPLST = $9C20 ; START OF DISP LIST
009B 1800 CR = $9B ; END OF LINE CHAR
1810 ;
0000 1820 *= $9000 ; PROGRAM STARTS AT
1830 ; 9000(HEX)
9000 1840 ARUN = * ; DEFINES A LABEL
1850 ; AT THIS LOCATION
9000 4C0690 1660 JMP CHGVEC ; CHANGE THE SYSTEM
1870 ; INIT VECTOR
1880 ;
9003 1890 AINIT = * ; DEFINES A LABEL
1900 ; AT THIS LOCATION
9003 4C2B90 1910 JMP CINIT ; INITIALISE C.A.
1920 ; SIMULATOR
1930 ; CHGVEC STEALS THE
1940 ; SYSTEM INIT VECTOR
1950 ; STORED AT LOC DOSINI
1960 ; SO THAT IF A <RESET> IS EXECUTED BOTH THE
1970 ; SYSTEM AND SIMULATOR ARE INITIALIZED
1980 ;
9005 1990 CHGVEC = * ;
9006 A50C 2000 LDA DOSINI ; GET LOW BYTE OF
9008 8D1990 2010 STA SAVINI ; SYS INIT VEC AND
2020 ; SAVE IT
2030 ; SAME FOR HIGH BYTE
9008 A50D 2040 LDA DOSINI+1 ;
900D 8D1A90 2050 STA SAVINI+1
2060 ; NOW GET LO BYTE
9010 A91E 2070 LDA #INIT&LO ; OF ADR OF NEW
9012 850C 2080 STA DOSINI ; INIT ROUTINE
2090 ; AND CHANGE LOW
2100 ; BYTE OF O.S INIT
LDA #INIT/HI ;ROUTINE
STA DOSINI+1 ;STORE RESULT
RTS ;END OF CHGVEC

SAVINI * = *+2 ;SAVE 2 LOCS FOR
FOR SAVINI

JMP (SAVINI) ;GOTO DOS INIT
THE NEXT SUBROUTINE
IS THE NEW INITIAL-
ISATION ROUTINE FOR
THE SYSTEM

INIT = *
JSR SYINIT ;DO SYSTEM INIT
JSR CINIT ;DO AUTOMATA INIT
RTS ;END OF INIT

REBOOT = *
GOTO COLDBOOT

JMP COLDSV ;ROUTINE
CINIT RESERVES MEMORY
INITS SCREEN DISPLAY
MODE AND INITS BKGND
CONTROL PROCESS

SNAME = *
NAME OF SCRN DEV

.CBYTE "S: ",CR ;

LDA #$70
STA RAMTOP
STA RAMSIZ
RESERVE MEMORY
NOW SET THE SCREEN DISP MODE
BY USING THE OPERATING SYSTEM
CENTRAL I/O (CIO) FACILITY

LDA #2
SET UP ACC FOR
A MODE 2 SCREEN

LDX #$60
SET UP INDEX REG
X TO DESIGNATE
IOCB#6

STA ICAX2,X
PASS INFO TO CIO

LDA #OPEN
SETUP FOR OPEN
OPEN SCREEN
DEVICE BY PASSING
INFO TO CIO

LDA #SNAME&LO

LDA ICBAH,X ;SET UP BUFFER
2640 ;POINTER TO POINT TO
2650 ;FILE NAME OF SCREEN
2660 ;DEVICE
2670 JSR CIOV ;GOTO CIO ENTRY
2680 BMI REBOOT ;POINT. N-BIT IS
2690 ;SET IF I/O REQUEST
2700 ;FAILED
2710 LDA #CLOSE ;IF I/O DONE
2720 STA ICCOM,X ;CLOSE SCREEN
2730 JSR CIOV ;BY USING CIO
2740 BMI REBOOT ;COLDBOOT IF
2750 ;ERROR ELSE CONTINUE
2760 :
2770 :
2780 ;A DISPLAY LIST IS A SET OF INS-
2790 ;TO THE ANTIC VIDEO PROCESOR
2800 ;ON POWER-UP THE OPERATING SYS
2810 ;TEM SETS THE SCREEN DISPLAY
2820 ;MODE TO MODE 2 AND THE START
2830 ;OF THE DISPLAY LIST IS PLACED
2840 ;AT 9C20 AND THE DATA FOR THE
2850 ;DISPLAY LIST IS PLACED AT 9C40
2860 ;HOWEVER, AFTER THE OPENING AND
2870 ;CLOSING OF THE SCREEN DEVICE
2880 ;AFTER RESERVING MEMORY IN THE
2890 ;PRECEDING ROUTINE, ANOTHER DISP-
2900 ;LAY LIST IS CREATED IN LOWER
2910 ;MEMORY BY THE O.S. THAT NEW
2920 ;DISPLAY LIST IS THE ONE POINTED
2930 ;TO BY THE O.S. POINTER DLIST.
2940 ;WE CAN SWITCH TO THE DISPLAY
2950 ;LIST THAT WE NOW CREATE FOR
2960 ;LATER USE BY MAKING DLIST POINT
2970 ;TO 9C20
2980 ;WE CHANGE THE MODE TO MODE 4
2990 ;BY CHANGING SOME PARAMETERS
3000 ;IN THE DISPLAY LIST THAT WAS
3010 ;CREATED BY O.S. BUT IS NOT
3020 ;BEING USED
3030 LDA #$44 ;LOAD MEMORY SCAN
3040 STA DSPLST+3 ;INSTRUCTION TO
3050 ;ANTIC, MODE 4
3060 ;
3070 LDX #23
3080 LDA #4 ;CHANGE THE 24
3090 ;LOCATIONS IN
3100 ;THE DISPLAY LIST
3110 ;STARTING AT DSPLST+5
3120 ;TO MODE 4
3130 DLOOP = *
3140 STA DSPLST+5,X
3150 DEX
3160 BNE DLOOP
3170 :
3180 ;NOW WE INITIALISE THE BACK-
3190 ;GROUND CONTROL PROCESS WHICH IS
; CALLED AS A TIMED-INTERRUPT
3210: ROUTINE
3220:
3230: THE BACKGROUND PROCESS SENDS
3240: A COMMAND TO THE SIMULATOR TO
3250: PERFORM 1 OF ITS 8 HARD WIRED
3260: FUNCTIONS BY SENDING THE
3270: FNMODE COMMAND EVERY TIME THE
3280: ROUTINE IS CALLED AS A RESULT
3290: OF A TIMED-INTERRUPT
3300:
3310: EVERY 1/60 SECOND TIMER CDTMV2
3320: IS DECREMENTED IF NON ZERO. IF
3330: AFTER DECREMENTING THE COUNT
3340: REACHES ZERO THIS ROUTINE IS
3350: CALLED. NOTE: LOCATIONS NUMFNC
3360: AND PERIOD SHOULD BE INITIAL-
3370: ISED IN BASIC BEFORE THE TIMER
3380: IS INITIALISED IN BASIC.
3390:

9066 78 3400 SEI ; DISABLE INT'S
9067 A975 3410 LDA #INTPT&LO
9069 8D2802 3420 STA CDTMA2 ; SET VECTOR ADRS
906C A990 3430 LDA #INTPT/HI
906E 8D2902 3440 STA CDTMA2+1 ; FOR TIMER CDTMV2
9071 58 3450 CLI ; ENABLE INT'S
9072 4C8E90 3460 JMP OVER ; JUST Initialise
9074 3470 CLI ; DON'T EXECUTE
9075 3480 JMP OVER ; UNTIL CALLED
9076 3490 LDA INTPT = * ; CDTMV2 INT'S GET
9077 3500 LDA NUMFNC ; VECTORED HERE
9078 8D9190 3510 STA SAVFNC ; GET AND SAVE
907B 3520 STA CDTMV2 ; FUNCTION # TO BE
907E 3530 LDA SAVFNC ; SENT TO SIM
9080 3540 JMP CLOOP ; AND TRY AGAIN
9084 4C7B90 3550 CMP I = * ; END OF INIT
9088 3560 LDA PERIOD ; SAVE LOCATIONS
908B 8D1A02 3570 STA CDTMV2 ; PERIOD,NUMFNC
908E 3580 RTS ; AND SAVFNC
9091 3590 CMP I = *+1 ; THE FOLLOWING ROUTINES ARE FOR
9094 35A0 3600 PERIOD *= *+1 ; TRANSFERRING DATA TO THE SIMU-
; LATOR. THE ROUTINE COMMND SENDS
; 1 OF 4 COMMANDS TO THE SIM WHEN
; IT IS CALLED. THE COMMANDS ARE:

; COMMAND EQUATES

0000 3820 WMODE = $00        ; WRITE TO DISPLAY
0010 3830 RMODE = $10        ; READ FROM DISP
0020 3840 TMODE = $20        ; SEND TRANSITION
0030 3860 ; FUNCTION TO SIM
0030 3860 FNMODE = $30        ; PERFORM 1 OF 8

; THE ROUTINE COMMND ASSUMES THAT
; THE COMMAND TO BE SENT TO THE
; SIMULATOR IS STORED IN THE LOW
; ORDER SIX BITS OF THE ACC
; THE TOP NIBBLE IS JUST 1 OF THE
; ABOVE 4 COMMANDS AND THE LOW
; NIBBLE SPECIFIES TO WHICH ONE
; OF EIGHT MEMORY PLANES OR
; SIMULATOR FUNCTIONS THE COMMAND
; APPLIES TO.

; COMMND INITIALISES THE 6820
; PARALLEL INTERFACE ADAPTOR
; (PIA) SO THAT COMMANDS CAN BE
; SENT. COMMND IS CALLED AS A
; SUBROUTINE BY SETDAT WHICH SETS
; UP DATA TRANSFER

; PIA EQUATES

D300 4060 PORTA = $D300        ; PORT A 6820
D301 4070 PORTB = $D301        ; PORT B 6820
D302 4080 PACTL = $D302        ; PORT A CONTROL REG
D303 4090 PBCTL = $D303        ; PORT B CONTROL REG
4100 ;

9092 4110 COMMND = *

9092 8D3C79 4120 STA SAVCMD    ; SAVE COMMAND
4130 ;

9095 AD00D3 4140 LDA PORTA    ;
9098 0940 4150 ORA #$40        ; MAKE SURE COMMND
909A 8D300 4160 STA PORTA     ; STROBE IS HIGH
4170 ;

909D A938 4180 LDA #$38        ; SETUP TO WRITE
909F 8D02D3 4190 STA PACTL     ; TO DATA DIRECT-
4200 ; ION CONTROL REG
4210 ; OF PORT A (6820)
4220 ;

90A2 8D03D3 4230 STA PBCTL     ; THIS SETS THE
4240 ; CB2 LINE ON THE
4250 ; 6820 TO LOW

90A5 A97F 4260 LDA #$7F        ; SETUP DIRECTION
90A7 8D02D3 4270 STA PORTA     ; FOR DATA LINES
4280 ; D7=IN D0-D6=OUT

90AA A93C 4290 LDA #$3C        ; SETUP TO ADDRESS
90AC 8D02D3 4300 STA PACTL     ; PORT A REGISTER
4310 ;
LDA PORTA ;READ PORT A TO
BMI RETURN ;CHECK IF ACK
RTS ;DONE

LDA SAVCMD ;RETRIEVE COMMAND
ORA #$40 ;SET STROBE BIT
STA PORTA ;SEND COMMAND
AND #$3F ;CLEAR STROBE BIT
STA PORTA ;AND MSB,SEND CMD
ORA #$40 ;SET STROBE BIT
STA PORTA ;MSB IS STILL 0
SEND COMMAND
ON EXIT IF COMMAND
WAS SENT N-BIT IS
CLEAR,ELSE IT IS

RETURN = * ;SET
RTS ;END OF COMMAND

SAVCMD *= *+1 ;SAVE LOC,SAVCMD

THE ROUTINE SETDAT SETS UP
DATA TRANSFERS BETWEEN THE
ATARI AND THE SIMULATOR. THE
ROUTINE SENDS A COMMAND BY
CALLING THE ROUTINE COMMND
AND THEN SETS THE DIRECTION OF
DATA TRANSFER BY WRITING TO
PORT B OF THE PIA.
THIS ROUTINE IS CALLED FROM
BOTH BASIC AND OTHER ASSEMBLY
LANGUAGE ROUTINES. THUS, IT HAS
TWO ENTRY POINTS.
A BASIC CALL TO A ASSEMBLY
LANGUAGE ROUTINE PLACES THE
ARGUMENTS OF THE CALL ONTO THE
STACK PRECEDED BY THE # OF ARGS

SETDAT = * ;BASIC ENTRY PT
PLA ;POP # OF ARGS
FROM BASIC CALL

PLA ;GET COMMAND
STA TEMP ;SAVE COMMAND
PLA

PLA ;GET DIRECTION OF
DATA TRANSFER
00=IN FF=OUT
TAX ;SAVE DIRECTION
IN X REG

LDA TEMP ;RETRIEVE COMMAND
4880 ;THE X REG HOLDS THE DIRECTION
4890 :
90D4   9000 SUPDAT = *
90D4 8DED90 4010 STA TEMP ;SAVE COMMAND
4920 ;AS SUBROUTINE
4930 ;CALL USES ACC
90D7   4040 RPTCMD = *
90D7 ADED90 4950 LDA TEMP ;RETRIEVE COMMAND
90DA 209290 4960 JSR COMMD ;SEND COMMAND
90DD 30F8 4970 BMI RPTCMD ;IF SIM WAS NOT
4980 ;READY SEND AGAIN
90DF   4990 WAIT = *
90DF AD00D3 5000 LDA PORTA ;WAIT FOR ACKN
90E2 30FB 5010 BMI WAIT ;FROM SIM,MSB=1
90E4 8E01D3 5020 STX PORTB ;SETUP DIRECTION
5030 ;OF DATA TRANSFER
5040 ;
90E7 A92C 5040 LDA #$2C ;WRITING THIS VAL
90E9 8D03D3 5050 STA PBCTL ;TO PORT B CON-
5060 ;TROL REGISTER
5070 ;PULLS THE CB2 LINE OF
5075 ;THE 6820 LOW ON THE
5080 ;RISING EDGE OF THE
5090 ;FIRST ENABLE PULSE
5095 ;FOLLOWING MPU WRITE
5100 ;TO PORTB DATA REG
5110 ;CB2 IS THEN SET ON
5115 ;THE RISING EDGE OF
5120 ;THE NEXT ENABLE (CLK)
5130 ;PULSE OF THE 6820
5140 ;THUS, A STROBE TO THE
5145 ;SIMULATOR IS GENER-
5150 ;ATED WHEN DATA IS
5160 ;WRITTEN TO PORT B
5165 ;
5170 ;
90EC 60 5180 RTS ;READY TO SEND
5190 ;DATA THROUGH
5200 ;PORT B DATA REGS
90ED 5210 TEMP *= *+1 ;SAVE LOC,TEMP
5220 :
5230 ;THE FOLLOWING ROUTINE SHOULD
5240 ;BE CALLED AFTER DATA TRANSFER
5250 ;IS FINISHED. IT SETS THE CB2
5260 ;LINE HIGH DISABLING FURTHER
5270 ;DATA TRANSFER AND SETS PORT B
5280 ;FOR INPUT AND SENDS A PERFORM
5290 ;A FUNCTION COMMAND
5300 ;THIS ROUTINE ALSO HAS TWO ENTRY
5310 ;POINTS ONE FOR BASIC AND ONE
5320 ;OTHER ASSEMBLY LANGUAGE ROUTINES
5330 ;
90EE 5340 STOPD = * ;BASIC ENTRY PNT
90EE 68 5350 PLA ;POP # OF ARGS
5360 :
5370 ;HERE IS THE ASSEMBLY
5380 ;LANGUAGE ENTRY
5390 ;POINT
90EF 5400 STPDAT = * ;STOP DATA TRNSFR
90EF A938 5410 LDA #$38 ;SETUP TO ADDRESS
90F1 8D03D3 5420 STA PBCTL ;PORT B DIRECTION
5430 ;CONTROL REGISTER
90F4 A900 5440 LDA #0 ;DIRECTION=INPUT
90F6 8D01D3 5450 STA PORTB
90F9 A930 5460 LDA #FNMODE ;PREPARE FNC CMMD
90FB 0940 5470 ORA #$40 ;SET BIT 6 HIGH
5480 ;TO STROBE SIM
90FD 8D00D3 5490 STA PORTA ;SEND COMMAND
9100 60 5500 RTS ;END OF STPDAT
5510 ;
5520 ;THE FOLLOWING ROUTINE WRITES
5530 ;THE CONTENTS OF THE DISPLAY
5540 ;BUFFER IN RAM TO A MEMORY PLANE
5550 ;OF THE SIMULATOR. THE MEMORY
5560 ;PLANE IS DISPLAYED ON THE
5570 ;COLOR MONITOR OF THE SIMULATOR
5580 ;BY THE DOT-TIMING LOGIC
5590 ;HARDWARE OF THE SIMULATOR
5600 :
5610 ;THIS ROUTINE ALSO HAS 2 ENTRY
5620 ;POINTS
5630 :
9101 5640 WRITED = * ;BASIC ENTRY PNT
9101 68 5650 PLA ;POP OFF # ARGS
9102 68 5660 PLA
9103 68 5670 PLA ;GET # OF MEMORY
5680 ;PLANE TO WRITE
5690 ;TO
9104 5700 WRTDSP = * ;ASSEMBLY ENT PNT
5710 ;THE ACC SHOULD
5720 ;CONTAIN THE NUMBER
5725 ;OF THE MEMORY PLANE
5730 ;TO BE WRITTEN TO
5740 ;BEFORE THIS ROUTINE
5745 ;IS CALLED
9104 18 5750 CLC ;CLEAR CARRY
9105 6900 5760 ADC #WMODE ;PREPARE WRITE
5770 ;TO PLANE CMMD
9107 8D3291 5780 STA WCMMD ;AND SAVE IT
910A A900 5790 LDA #DBUF&LO
910C 85CB 5800 STA PGZ ;STORE START OF
910E A970 5810 LDA #DBUF/HI
9110 85CC 5820 STA PGZ+1 ;DISPLAY BUFFER
5830 ;IN A PAGE ZERO
5840 ;LOCATION
5850 ;PGZ AND PGZ+1 WILL
5860 ;BE USED AS POINTERS
5870 ;INTO THE DISPLAY
5880 ;BUFFER
9112 5890 WRTAGN = *
9112 A03291 5900 LDA WCMMD ;RETRIEVE COMMAND
9115 A2FF 5910 LDX #OUT ;DATA TRANSFER
5920 ;DIRECTION IS OUT
9117 20D490 5930 JSR SUPDAT ;SETUP DATA TRANS
05940 3A0E0 BMI WRTAGN ; FER FOR WRITE
05950 30F6 30F6 ; IF SIM WAS NOT
05960 31AB 31AB ; READY SEND AGAIN
05970 3120 3120 ; ELSE START SENDING
05980 3121 3121 ; DATA THROUGH PORT B

05990 8D01D3 30F6 STA PORTB ; USE INDEX REG Y
06000 6030 6030 INC Y ; TO SEND THE NEXT
06010 6040 6040 BNE WLOOP ; 256 BYTES OF THE
06020 6050 6050 ; DISPLAY BUFFER
06030 6060 6060 INC PGZ +1 ; INCREMENT THE
06040 6070 6070 ; HIGH BYTE OF
06050 6080 6080 ; POINTER INTO BUF
06060 6090 6090 ; SINCE 256 B DONE
06070 6100 6100 LDA PGZ +1 ; CHECK IF END OF
06080 6110 6110 CMP #DBUF/HI +32
06090 6120 6120 ; BUFFER. WE ARE
060A0 6130 6130 ; DONE WHEN WE HAVE
060B0 6140 6140 ; SENT 8K BYTES

06100 6150 6150 BNE WLOOP
06110 6160 6160 JSR STPDAT ; STOP DATA TRNSFR
06120 6170 6170 RTS

06130 6180 6180 WCMMD *= +1 ; SAVE LOC.WCMMD

06190 6200 6200 ; THE FOLLOWING ROUTINE IS ANA-
06210 6220 6220 ; LOGOUS TO WRTDSP EXCEPT THAT
06230 6230 6230 ; IT PERFORMS A READ OPERATION
06240 6240 6240 ; FROM THE SIM MEMORY PLANES
06250 6250 6250 ;

06260 6260 6260 RDDSP *= * ; BASIC ENT PT
06270 6270 6270 PLA ; POP # ARGS
06280 6280 6280 PLA
06290 6290 6290 PLA ; GET # OF MEMORY
06300 6300 6300 ; MEMORY PLANE TO READ

06310 6310 6310 RDDPLY *= * ; ASSEMBLY ENT PNT
06320 6320 6320 ; THE ACC SHOULD
06330 6330 6330 ; HOLD THE # OF THE
06340 6340 6340 ; MEM PLANE TO READ

06350 6350 6350 CLC ; CLEAR CARRY

06360 6360 6360 ADC #RMODE ; PREPARE A READ
06370 6370 6370 STA RCMMD ; FROM MEM PLANE

06380 6380 6380 ; CMMD AND SAVE IT

06390 6390 6390 LDA #DBUF&LO
063A0 6400 6400 STA PGZ ; STORE START OF

06400 6410 6410 LDA #DBUF/HI

06420 6420 6420 STA PGZ+1 ; DISPLAY BUFFER

06430 6430 6430 ; IN PAGE 0 LOCS

06440 6440 6440 ; PGZ AND PGZ+1 WILL

06450 6450 6450 ; BE USED AS POINTERS

06460 6460 6460 RTRCMD *= * ; INTO DISPLAY BUFFER

06470 6470 6470 LDA RCMMD ; RETRIEVE COMMAND

06480 6480 6480 LDX #IN ; DIRECTION FOR

06490 6490 6490 ; TRANSFER IS INPUT
9149 20D490 6500 JSR SUPDAT ;SET UP DATA TNSFR
914C 30F6 6510 BMI RTRCMD ;IF SIM WAS NOT
914E A000 6520 ;READY SEND AGAIN
9150 AD01D3 6530 RLOOP = *
9150 AD01D3 6550 LDA PORTB ;READ DATA FROM
9153 8D01D3 6560 STA PORTB ;MEMORY PLANE AND
9155 91CB 6570 ;WRITE BACK TO
9156 91CB 6580 ;PORT B TO PRODUCE
9158 8D01D3 6590 ;STROBE. REFER TO
915A 91CB 6600 ;COMMENTS FOR SUPDAT
915C 91CB 6610 STA (PGZ),Y ;STORE DATA FROM
915E CB 6620 INY ;MEM PLANE INTO
9160 D0F5 6630 BNE RLOOP ;NEXT 256 LOCS OF
9162 8D01D3 6640 ;DISPLAY BUFFER
9164 91CB 6650 INC PGZ+1 ;INCREMENT HIGH
9166 8D01D3 6660 ;BYTE OF POINTER
9168 91CB 6670 ;SINCE 256 B READ
916A 91CB 6680 LDA PGZ+1 ;CHECK IF END OF
916C CB 6690 CMP #DBUF/HI+32
916E D0ED 6700 BNE RLOOP ;BUFFER. WE ARE
9170 20EF90 6710 JSR STPDAT ;DONE WHEN WE HAVE
9172 8D01D3 6720 ;READ 8K BYTES
9174 60 6730 RTS
9176 70EF90 6740 ;
9178 6E90 6750 RCMMD *= *+1 ;SAVE LOC,RCMMD
917A 70EF90 6760 ;
917C 70EF90 6770 ; THE FOLLOWING ROUTINE WHEN
917E 70EF90 6780 ; CALLED AFTER AN "OPEN" (DEVICE)
9180 70EF90 6790 ; COMMAND IN BASIC WILL PERFORM
9182 70EF90 6800 ; DATA TRANSFER FROM THE DISK
9184 70EF90 6810 ; DRIVE TO THE DISPLAY BUFFER
9186 70EF90 6820 ; (LOADBF), OR SAVE THE DISPLAY
9188 70EF90 6830 ; BUFFER ON DISK (SAVBUF).
918A 70EF90 6840 ; EACH ROUTINE USES THE CIO
918C 70EF90 6850 ; FACILITY OF THE OPERATING SYS
918E 70EF90 6860 ; TEM. TO USE CIO WE SET THE
9190 70EF90 6870 ; COMMAND BYTE, THE BUFFER ADDRESS
9192 70EF90 6880 ; AND THE BUFFER LENGTH. IN
9194 70EF90 6890 ; LOADBF THE COMMAND BYTE IS $07
9196 70EF90 6900 ; FOR "GET CHARACTERS" AND IN
9198 70EF90 6910 ; SAVBUF THE COMMAND IS $0B FOR
919A 70EF90 6920 ; "PUT CHARACTERS". IN BOTH
919C 70EF90 6930 ; ROUTINES THE BUFFER ADDRESS IS
919E 70EF90 6940 ; THE STARTING ADDRESS OF THE
91A0 70EF90 6950 ; DISPLAY AND THE BUFFER LENGTH
91A2 70EF90 6960 ; PARAMETER IS THE LENGTH OF THE
91A4 70EF90 6970 ; BUFFER.
91A6 70EF90 6990 ;
91A8 70EF90 69A0 ; BOTH ROUTINES EXPECT THE
91AA 70EF90 69B0 ; IOCB # TO BE PASSED AS AN ARGU-
91B2 70EF90 69C0 ; MENT OF THE BASIC CALL TO THEM
91B4 70EF90 69D0 ;
91B6 70EF90 7050 LOADBF = * ; BASIC ENTRY PNT
9168 A007 7060 7070 7080
916A 7090 COMMON = * ;LOAD DISP BUFFER
916A 7100 ;GET CHAR CMMD
916A 7110 ;;THIS PART OF THE
916A 7120 ;CODE IS COMMON
916A 7130 PLA ;TO BOTH LOADBF
916A 7140 PLA ;SAVEBUF
916A 7150 PLA ;POP # OF ARGS
916A 7160 ASL A ;GET IOCNUMBER
916A 7170 ASL A
916A 7180 ASL A
916A 7190 ASL A
916A 7200 TAX
916A 7210 TYA
916A 7220 STA IC[COM,X
916A 7230 LDA #DBUF&LO
916A 7240 STA IC[BAAL,X
916A 7250 LDA #DBUF/HI
916A 7260 STA IC[BAH,X
916A 7270 LDA #0
916A 7280 STA IC[BLL,X
916A 7290 LDA #$20
916A 7300 STA IC[BLH,X
916A 7310 JSR CIOV
916A 7320 STY RESLTL
916A 7330 LDA #0
916A 7340 STA RESLTH
916A 7350 ; LOW THEN HIGH
916A 7360 RTS
916A 7370
916A 7380 SAVBUF = *
916A 7390 ;BASIC ENTRY PNT
916A 7400 LDY #PUTC
916A 7410 JMP COMMON
916A 7420 ;PUT CHAR CMMD
916A 7430 ;GO TO COMMON
916A 7440 ;PART OF DISK
916A 7450 SETPLN = *
916A 7460 ;INITS THE MEM
916A 7470 ;PLANE WITH THE
916A 7480 ;VALUE PASSED TO THE
916A 7490 PLA
916A 7500 PLA
916A 7510 PLA
916A 7520 TAY
916A 7530 PLA
916A 7540 PLA
916A 7550 SPLN = *
916A 7560 ;ASSEMBLY ENTR PNT
916A 7570 ;THE Y REG SHOULD
916A 7580 ;HOLD THE VALUE TO
916A 7590 ;WRITE AND THE ACC
916A 7600 STY VAL
916A 7610 LDX #OUT
916A 7620 ;DATA DIR = OUT
91A4 18 7620  CLC
91A5 6900 7630  ADC  #WMODE  ;FORM WRITE TO
7640  ;PLANE COMMAND
91A7 20D490 7650  JSR  SUPDAT  ;SETUP FOR DATA
7660  ;TRANSFER
91AA A020 7670  LDY  #$20  ;WANT TO WRITE
91AC A200 7680  LDX  #0  ;8K TIMES
91AE ADDE91 7690  LDA  VAL  ;RETRIEVE VALUE
91B1 7700 STLOOP  =  *  ;SEND TO MEM PLN
91B1 8D01D3 7710  STA  PORTB
91B4 CA 7720  DEX
91B5 D0FA 7730  BNE  STLOOP  ;KEEP ON SENDING
91B7 88 7740  DEY  ;UNTIL ALL 8K
91B8 D0F7 7750  BNE  STLOOP  ;DONE
91BA 20EF90 7760  JSR  STPDAT  ;STOP DATA TRNSFR
91BD 60 7770  RTS
7780
91BE 7790 VAL  *=  *+1  ;SAVE LOC FOR VAL
7800  ;
91BF 7810 SETBUF  =  *  ;SETS DISPLAY BFR
7820  ;TO VALUE PASSED
7830  ;TO ROUTINE
91BF 88 7840  PLA  ;POP # ARGS
91C0 88 7850  PLA  ;GET VALUE TO
91C1 88 7860  PLA  ;WRITE TO DSP BFR
91C2 7870 STBUF  =  *  ;ASSEMBLY ENT PNT
7880  ;ACC SHOULD HOLD
7890  ;VALUE TO WRITE
91C2 8DE391 7900  STA  BUFVAL  ;SAVE VALUE TO
7910  ;WRITE
7920  ;NOW STORE START OF
7930  ;BUFFER IN LOCS PGZ
91C5 A900 7940  LDA  #DBUF&LO  ;AND PGZ+1
91C7 85CB 7950  STA  PGZ  ;LOW BYTE
91C9 A970 7960  LDA  #DBUF/HI
91CB 85CC 7970  STA  PGZ+1  ;THEN HIGH BYTE
91CD A000 7980  LDY  #0  ;READY INDEX REG
91CF 7990 BLOOP  =  *  ;RETREIVE VALUE
91CF ADE391 8000  LDA  BUFVAL  ;TO WRITE TO BUFR
8010  ;SEND VAL TO BUFR
91D2 91CB 8020  STA  (PGZ),Y
91D4 C8 8030  INY  ;UNTIL END OF THE
91D5 D0F8 8040  BNE  BLOOP  ;BUFFER=8K WRITES
91D7 E6CC 8050  INC  PGZ+1  ;INCREMENT HIGH
8060  ;BYTE OF BFR PNTR
8070  ;SINCE 256 B DONE
91D9 A5CC 8080  LDA  PGZ+1  ;CHECK IF 8K DONE
91DB C990 8090  CMP  #DBUF/HI+32
91DD D0F0 8100  BNE  BLOOP  ;IF NOT CONTINUE
8110  ;WRITING DATA
91DF 20EF90 8120  JSR  STPDAT  ;ELSE STOP
91E2 60 8130  RTS
8140
91E3 8150 BUFVAL  *=  *+1  ;SAVE LOC,BUFVAL
8160
91E4 8170 JMPTAB  =  *  ;JUMP TABLE FOR
Routines which and, or, exor the disp buffer with a memory plane.

The following routine and's, or's, exor's the disp buffer with a memory plane.

AND, OR

; INITSN = *

; EXOR THE DISPLAY BUFFER WITH THE PLANE SPECIFIED

; POP # ARG'S

; GET # OF FUNC

; AND SAVE IT

; GET PLANE # TO USE WITH BUFFER

; ASSEMBLY ENT PNT

; ACC SHOULD HOLD # OF PLN TO EXOR

; LOW BYTE

; THEN HIGH BYTE

; GET ADDRESS FROM JUMP TABLE

; RETURN FROM LOGICAL OPERATION

; STORE EXOR'ED VALUE IN BUFFER

; DO FOR NEXT 256 LOC'S IN BUFR

ANDP

; ROUTINES WHICH

ORP ; AND, OR, EXOR THE

XORP ; DISP BUFFER WITH

A MEMORY PLANE

THE FOLLOWING ROUTINE AND'S,

OR'S, EXOR'S THE DISP BUFFER

WITH A MEMORY PLANE

AND, OR

; INITSN = *

; EXOR THE DISPLAY BUFFER WITH THE PLANE SPECIFIED

; POP # ARG'S

; GET # OF FUNC

; AND SAVE IT

; GET PLANE # TO USE WITH BUFFER

; ASSEMBLY ENT PNT

; ACC SHOULD HOLD # OF PLN TO EXOR

; LOW BYTE

; THEN HIGH BYTE

; GET ADDRESS FROM JUMP TABLE

; RETURN FROM LOGICAL OPERATION

; STORE EXOR'ED VALUE IN BUFFER

; DO FOR NEXT 256 LOC'S IN BUFR
INC PGZ+1 ; INCREMENT HIGH

LDA PGZ+1 ; CHECK IF END OF

CMP #DBUF/HI+32

BNE FNLOOP ; BUFFER

JSR STPDAT ; STOP IF DONE

RTS

ANDP = * ; AND WITH VALUE

AND PORTB ; FROM MEM PLANE

STA PORTB ; WRITE IT BACK TO

PORTB TO PRODUCE A STROBE TO

THE SIMUL SO THAT IT WILL SEND

THE NEXT VALUE

JMP FNRETN ; STORE IN BUFFER

ORP = * ; OR WITH VALUE

ORA PORTB ; FROM PLANE

STA PORTB ; CONTINUE

JMP FNRETN ; CONTINUE

XORP = * ; EXOR BUF VALUE

EOR PORTB ; WITH VALUE FROM

STA PORTB ; MEM PLANE

JMP FNRETN ; CONTINUE

OFFSET *= *+1 ; SAVE LOC, OFFSET

JMPADR *= *+2 ; SAVE 2 LOCS

.END
REM THIS BASIC PROGRAM CONTROLS THE CELLULAR AUTOMATA SIMULATOR BY
calling appropriate assembly language routines and provides the
user interface for use of the simulator.
REM THE PROGRAM WORKS IN THE FOLLOWING WAY: IT ACCESSES THE DISK
AND THE SCREEN A MENU OF COMMANDS
REM THAT IS AVAILABLE TO THE USER. THE USER THEN ENTERS THE
REM SELECTION DESIRED BY ENTERING A SINGLE KEY. THIS CAUSES A
REM JUMP TO THE ROUTINE WHICH HANDLES THAT FUNCTION.
REM THE MENU IS READ IN FROM DISK INSTEAD OF USING PRINT STATEMENTS
REM BECAUSE THERE IS A SHORTAGE OF RAM
REM IN FACT MUCH OF THE COMMENTS FOR THIS PROGRAM WERE
REM AFTER THE PROGRAM WAS TESTED. THE EXECUTABLE CODE OF BOTH
REM PROGRAMS ARE IDENTICAL
REM OPERATING SYSTEM EQUATES
KCODE=764 REM LOC CONTAINS CODE FOR LAST KEY Pressed
ERRSV=195 REM LOC CONTAINS ERROR NUMBER
STOPLN=186 REM LOC CONTAINS LINE AT WHICH STOP OCCURRED
CDTMV2=538 REM COUNT REGISTER OF COUNTDOWN TIMER1
PORTB=54017 REM PORTB OF 6820 PIA
REM THE FOLLOWING ARE THE CODE VALUES FOR VARIOUS KEYS
CLR=256 REM IF PEEK(KCODE)=255 THEN NO KEY INPUT
A=63 B=61 C=18 D=58 E=42
F=56 G=61 H=57 I=13 J=1
K=5 L=0 M=37 N=35 O=8
P=10 Q=47 R=40 Y=43
REM EQUATES FOR ASSEMBLY LANGUAGE SUBROUTINES
DISPBUF=28672 REM START OF DISPLAY BUFFER =7000 HEX
RDOSP=37171 REM READS SIMULATOR MEM PLANE TO DISP BUFFER
SAVBUF=37268 REM SAVES DISP BUFFER ON DISK
WRITED=37121 REM WRITES DISP BUFFER TO SIM MEMORY PLANE
LODBF=37224 REM LOADS DISP BUFFER FROM DISK
SETDAT=37064 REM SETS UP DATA TRANSFER FROM ATARI TO SIM
STPDAT=37102 REM DISABLES DATA TRANSFER FROM ATARI TO SIM
SETPLN=37273 REM SETS PLANE WITH VALUE PASSED TO IT
SETBUF=37311 REM SETS DISPLAY BUFFER W/VALUE PASSED TO IT
BUFPLN=37354 REM AND'S, EXOR'S, OR'S DISP BUFFER W/MEM PLANE
PERIOD=37007 REM LOCATION CONTROLS SPEED OF SIMULATOR
NUMFNC=37008 REM LOCATION CONTROLS WHAT SIMULATOR IS DOING
REM
DIM CS(1) REM USED FOR CLEARING SCREEN
DIM DS(40) REM USED FOR USER SUPPLIED FILE NAMES FOR
DIM TS(40) REM DISK DRIVE ACCESSES
DIM FNAMES$(20) REM USED TO ACCESS A TEXT FILE FROM DISK
DIM TXTS(80) REM USED TO GIVE USER-PROMPTS
DIM SAVES$(128) REM USED TO SAVE TRANSITION TABLES
REM *
REM * START OF MAIN LOOP OF PROGRAM *
REM OPEN #1,8,0,"E:" REM OPEN SCREEN FOR OUTPUT
REM ALL FURTHER PRINT STATEMENTS OF THE FORM "PRINT #1:"<...>"
REM WILL BE PRINTED ON THE SCREEN
0467 PRINT #1;"CS" :REM CLEAR SCREEN
0470 REM READ IN THE TOP LEVEL USER-PROMPTS FROM DISK
0475 REM AND PRINT IT OUT TO THE SCREEN
0480 FNAME$="D:TOPVL.PMT"
0490 GOSUB 9000 :REM PROMPT USER
0500 POKE KCODE,CLR :REM CLEAR KEY CODE
0510 CHAR=PEEK(KCODE) :REM GET KEY CODE FOR LAST KEY PRESSED
0520 IF CHAR=CLR THEN GOTO 670 : REM WAIT UNTIL A KEY IS PRESSED
0525 REM * JUMP TABLE FOR TOP LEVEL *
0530 IF CHAR=A THEN GOTO 880
0540 IF CHAR=B THEN GOTO 1350
0550 IF CHAR=C THEN GOTO 1720
0560 IF CHAR=D THEN GOTO 2320
0570 IF CHAR=E THEN GOTO 2580
0580 IF CHAR=F THEN GOTO 2890
0590 IF CHAR=G THEN GOTO 7800.
0600 IF CHAR=H THEN GOTO 8200
0610 IF CHAR=I THEN GOTO 8800
0620 PRINT #1;"PLEASE ENTER A VALID COMMAND"
0630 GOTO 660 :REM INSIST ON VALID INPUT
0640 REM
0650 REM EACH OF THE HANDLERS FOR THE TOP LEVEL FUNCTIONS RESIDE
0660 REM IN LINES 880 TO 3150 FOR COMMANDS "A" THROUGH "F" AND
0670 REM ALSO IN LINES 7800 TO 8850 FOR COMMANDS "G" THROUGH "I"
0680 REM EACH HANDLER ALSO HAS ITS OWN INTERNAL JUMP TABLE TO
0690 REM HANDLE COMMANDS LOCAL TO THAT TOP LEVEL HANDLER
0700 REM
0710 REM *******************************************************
0720 REM HANDLER FOR DEFINING AND DOWNLOADING A TRANSITION RULE
0730 REM *******************************************************
0740 PRINT #1;"CS" :REM CLEAR SCREEN
0750 REM READ IN USER-PROMPTS FROM DISK
0760 FNAME$="D:TRNSTN.PMT"
0770 GOSUB 9000 :REM PROMPT USER
0780 POKE KCODE,CLR
0790 PRINT #1;"TYPE ANY CHAR TO CONTINUE:".
0800 CHAR=PEEK(KCODE) :REM WAIT UNTIL KEY IS PRESSED
0810 IF CHAR=CLR THEN GOTO 1030
0820 REM
0830 PRINT #1;"AFTER THE USER HAS INSERTED THE"
0840 PRINT #1;"CHANGES BY USING THE SCREEN EDITOR"
0850 PRINT #1;"THE COMMAND 'CONT' FOLLOWED BY <CR>"
0860 PRINT #1;"SHOULD BE ENTERED TO CONTINUE"
0870 LIST 30000,30010 :REM PRINT LINES 30000 TO 30010 OF THIS
0880 REM PROGRAM WHICH SHOULD DEFINE THE TRANSITION FUNCTION
0890 POKE KCODE,CLR
1000 STOP :REM GOTO SCREEN EDIT MODE
1010 REM WHEN USER TYPES "CONT" EXECUTION CONTINUES W/NEXT LINE
1020 PRINT #1;"TRANSITION RULE DEFINITION PROCESS"
1030 PRINT #1;"COMPLETED"
1040 PRINT #1;"ENTER 'Y' TO DOWNLOAD THIS RULE"
1050 PRINT #1;"ANY OTHER CHAR TO ABORT"
1060 POKE KCODE,CLR
1070 CHAR=PEEK(KCODE)
1180 IF CHAR=CLR THEN GOTO 1170
1190 IF CHAR>Y THEN GOTO 1220
1195 REM IF INPUT CHAR IS NOT "Y" THEN ABORT ELSE DOWNLOAD RULE
1200 GOSUB 3170 :REM DOWNLOAD ROUTINE
1210 GOTO 470 :REM GOTO MAIN LOOP OF PROGRAM
1215 REM
1220 PRINT #1;:CS :REM ABORT DOWNLOADING OF TRANSITION RULE
1230 PRINT #1;:"RULE NOT DOWNLOADED"
1240 POKE KCODE, CLR
1250 PRINT #1;:"ENTER ANY CHAR TO GET BACK TO TOP"
1260 PRINT #1;:"LEVEL OF THE PROGRAM"
1270 CHAR=PEEK(KCODE)
1280 IF CHAR=CLR THEN GOTO 1270
1290 GOTO 470 :REM ALSO RETURN TO MAIN LOOP OF PROGRAM
1300 REM * END OF TRANSITION RULE DEFINITION PROCESS *
1310 REM
1320 REM **********************************************************************
1330 REM HANDLER FOR SETTING UP THE INITIAL STATE OF THE SIMULATOR
1340 REM **********************************************************************
1350 PRINT #1;:CS :REM CLEAR SCREEN
1355 REM ACCESS USER PROMPTS FROM THE FILE "D:SETUPSIM.PMT"
1360 FNAMES="D:SETUPSIM.PMT"
1370 GOSUB 9000 :REM AND PROMPT USER
1380 POKE KCODE, CLR
1390 CHAR=PEEK(KCODE) :REM WAIT FOR USER INPUT
1400 IF CHAR=CLR THEN GOTO 1530
1410 REM JUMP TABLE FOR SETUP DISPLAY HANDLER
1420 IF CHAR=A THEN GOTO 3440
1430 IF CHAR=B THEN GOTO 3560
1440 IF CHAR=C THEN GOTO 3660
1450 IF CHAR=D THEN GOTO 3780
1460 IF CHAR=E THEN GOTO 3880
1470 IF CHAR=F THEN GOTO 3970
1480 IF CHAR=G THEN GOTO 4070
1490 IF CHAR=H THEN GOTO 4160
1500 IF CHAR=I THEN GOTO 4230
1510 IF CHAR=J THEN GOTO 4300
1520 IF CHAR=K THEN GOTO 4380
1530 PRINT #1;:"PLEASE ENTER A VALID COMMAND"
1540 GOTO 1520 :REM INSIST ON VALID INPUT
1550 REM * END OF SETUP SIMULATOR HANDLER *
1560 REM **********************************************************************
1570 REM RUN SIMULATOR ROUTINES ARE HANDLED FROM HERE
1580 REM **********************************************************************
1590 REM **********************************************************************
1600 PRINT #1;:CS :REM CLEAR SCREEN
1610 REM ACCESS USER-PROMPTS FROM DISK
1620 FNAMES="D:RUNSIM1.PMT"
1630 GOSUB 9000 :REM AND PROMPT USER
1640 POKE KCODE, CLR
1650 CHAR=PEEK(KCODE) :REM WAIT FOR INPUT
1660 IF CHAR=CLR THEN GOTO 1840
1670 IF CHAR=Q THEN GOTO 470 :REM USER WANTED TO RETURN TO MAIN LOOP
1680 REM IF INPUT CHAR WAS "Q" ELSE PROCEED
1690 PRINT #1;:CS :REM CLEAR SCREEN AGAIN
1700 FNAMES="D:RUNSIM2.PMT"
1710 GOSUB 9000 :REM AND GET USER-PROMPTS
2060 POSITION 2,19 :REM PLACE CURSOR ON LOWER-LEFT
2065 PRINT #1;"PLEASE ENTER YOUR COMMAND:"
2070 POKE KCODE,CLR :REM CLEAR KEY CODE
2080 CHAR=PEEK(KCODE) :REM AND WAIT FOR INPUT
2090 IF CHAR=CLR THEN GOTO 2080
2100 REM JUMP TABLE FOR RUNNING SIMULATOR
2110 IF CHAR=A THEN GOTO 4480
2120 IF CHAR=B THEN GOTO 4670
2130 IF CHAR=C THEN GOTO 4680
2140 IF CHAR=D THEN GOTO 4750
2150 IF CHAR=E THEN GOTO 4830
2160 IF CHAR=F THEN GOTO 4920
2170 IF CHAR=G THEN GOTO 5010
2180 IF CHAR=H THEN GOTO 5100
2190 IF CHAR=I THEN GOTO 5190
2200 IF CHAR=J THEN GOTO 5330
2210 IF CHAR=K THEN GOTO 5410
2220 IF CHAR=L THEN GOTO 5480
2240 IF CHAR=N THEN GOTO 5790
2250 IF CHAR=O THEN GOTO 470
2260 PRINT #1;"PLEASE ENTER A VALID COMMAND"
2270 GOTO 2070 :REM INSIST ON VALID INPUT
2280 REM * END OF RUN SIMULATOR HANDLER *
2290 REM ****************************************************
2300 REM START OF SAVE/RETRIEVE TRANS RULE HANDLER
2310 REM *****************************************************
2315 REM
2320 PRINT #1;CS :REM CLEAR SCREEN
2325 REM ACCESS USER-PROMPTS FROM DISK
2330 FNAME$="D:SVRVTVR.PMT"
2340 GOSUB 9000 :REM AND PRINT IT ON SCREEN
2410 POSITION 2.19 :PLACE CURSOR ON LOWER LEFT
2420 PRINT #1;"PLEASE ENTER YOUR COMMAND:"
2430 POKE KCODE,CLR :REM CLEAR KEY CODE
2440 CHAR=PEEK(KCODE) :REM AND WAIT FOR INPUT
2450 IF CHAR=CLR THEN GOTO 2440
2460 REM * JUMP TABLE *
2470 IF CHAR=A THEN GOTO 6530
2480 IF CHAR=B THEN GOTO 6680
2490 IF CHAR=C THEN GOTO 6840
2500 IF CHAR=D THEN GOTO 7000
2510 IF CHAR=E THEN GOTO 470
2520 POSITION 2,19 :REM KEEP SCREEN CLEAN BY OVERWRITING AT
2525 REM SAME PLACE ON THE SCREEN TO PREVENT SCROLLING
2530 PRINT #1;"PLEASE ENTER A VALID COMMAND:"
2540 GOTO 2430 :INSIST ON VALID INPUT
2545 REM * END OF SAVE/RETRIEVE TRANSITION RULE HANDLER *
2550 REM ***********************************************
2560 REM START OF SAVE/RETRIEVE DISPLAY BUFFER HANDLER
2570 REM ***********************************************
2580 PRINT #1;CS :REM CLEAR SCREEN
2585 REM AND ACCESS USER-PROMPTS FROM DISK
2590 FNAME$="D:SVRVTVD.PMT"
2600 GOSUB 9000 :REM PRINT PROMPTS ON DISK
2750 POSITION 2.19 :REM KEEP SCREEN CLEAN BY OVERWRITING
2760 PRINT #1;"PLEASE ENTER YOUR COMMAND:"

2770 POKE KCODE, CLR : REM AT THE SAME LOCATION
2780 CHAR = PEEK(KCODE) : REM CLEAR KEY CODE AND WAIT FOR INPUT
2790 IF CHAR = CLR THEN GOTO 2780
2800 IF CHAR = A THEN GOTO 7220
2810 IF CHAR = B THEN GOTO 7380
2820 IF CHAR = C THEN GOTO 470
2830 POSITION 2, 19 : REM KEEP SCREEN CLEAN
2840 PRINT #1; "PLEASE ENTER A VALID COMMAND:"
2850 GOTO 2770 : REM INSIST ON VALID INPUT

2855 REM * END OF SAVE/RETRIEVE DISPLAY BUFFER HANDLER *
2860 REM ***********************************************
2870 REM START OF SAVE/RETRIEVE MEMORY PLANE HANDLER
2880 REM ***********************************************
2890 PRINT #1; C$ : REM CLEAR SCREEN
2900 FNAMES = "D: SAVRTVMP.PMT"
2910 GOSUB 9000 : REM AND PRINT IT
2920 POSITION 2, 19 : REM PLACE CURSOR ON LOWER LEFT
2930 PRINT #1; "PLEASE ENTER THE COMMAND YOU WISH: "
2940 POKE KCODE, CLR : REM CLEAR KEY CODE
2950 CHAR = PEEK(KCODE) : REM AND WAIT FOR INPUT
2955 REM * JUMP TABLE *
2960 IF CHAR = CLR THEN GOTO 2950
2970 IF CHAR = A THEN GOTO 7530
2980 IF CHAR = B THEN GOTO 7630
2990 IF CHAR = C THEN GOTO 470
3000 POSITION 2, 19 : REM KEEP SCREEN CLEAN BY OVERWRITING
3010 PRINT #1; "PLEASE ENTER A VALID COMMAND:"
3020 GOTO 2940 : REM INSIST ON VALID INPUT
3030 REM * END OF TOP LEVEL HANDLERS FOR COMMANDS "A" THROUGH "F" *
3040 REM * MORE TOP LEVEL HANDLERS AT LINES 7800 TO 8850 *
3050 REM
3060 END : REM PREVENT TRIGGERING OF FOLLOWING SUB-HANDLERS
3070 REM
3080 REM DOWNLOAD ROUTINE IS HERE
3090 REM
3100 INDEX = 1
3110 REM CALL ASSEMBLY LANGUAGE ROUTINE TO SETUP DATA TRANSFER FROM
3120 REM ATARI TO SIMULATOR
3130 REM
3140 REM DISABLE DATA TRANSFER; THIS INITIALISATION KLUGE IS DUE TO
3150 REM THE FACT THAT THE DISK DRIVE ALSO USES LINE CB2 OF 6820 PIA
3160 REM REFER TO ASSEMBLY LANGUAGE LISTING
3170 REM ONLY TRANSITION TABLE 0 AVAILABLE
3180 REM
3190 X = USR(SETDAT, 32, 255) : REM 32+N=DOWNLOAD TRANSITION FUNCTION TO
3200 REM TRANSITION TABLE N OF SIMULATOR.
3210 X = USR(STPDAT) : REM 255=DIRECTION OF DATA TRANSFER IS OUTPUT
3220 REM REFER TO ASSEMBLY LANGUAGE LISTING
3230 REM THE TRANSITION TABLE IS A LOOK-UP TABLE THAT HOLDS THE NEXT
3240 REM VALUE OF THE CENTER CELL C FROM THE PRESENT STATE OF CENTER
3250 REM CELL AND ITS NEIGHBORING CELLS N, S, W, E WHICH ARE ORTHOGONAL
3260 REM TO THE CENTER CELL
3270 FOR C = 0 TO 1 : REM FOR EVERY POSSIBLE VALUE OF PRESENT STATE
3280 FOR W = 0 TO 1
3290 FOR E = 0 TO 1
3260 FOR N=0 TO 1
3270 FOR S=0 TO 1
3280 GOSUB 30000 :REM CALCULATE THE NEXT STATE X OF THE CENTER
3285 REM CELL AND WRITE IT TO THE SIMULATOR VIA PORTB
3287 REM TO FORM THE TRANSITION TABLE ONLY LSB USED
3290 POKE PORTB,X :REM X=0 OR 1
3300 POKE PORTB,0 :REM THIS SYNCs THE SIM AND VALUE IS IGNORED
3305 REM NOW SAVE THE RETURNED VALUE IN A STRING
3310 LET SAVE$(INDEX)=CHR$(X)
3320 INDEX=INDEX+1 :REM INCREMENT INDEX INTO STRING
3330 NEXT S
3340 NEXT N
3350 NEXT E
3360 NEXT W
3370 NEXT C :REM DO FOR ALL POSSIBLE CASES
3380 X=USR(STPDAT) :REM DISABLE DATA TRANSFER
3382 REM SINCE THIS ROUTINE CHANGES GLOBAL KCODE VARIABLES
3383 WE REINITIALISE THEM BEFORE RETURNING
3384 N=35:C=18:E=42
3390 RETURN
3400 REM ** END OF DOWNLOAD **
3410 REM ******************************************************
3420 REM HANDLER TO WRITE A VALUE TO THE SIMULATOR MEM PLANES
3430 REM ******************************************************
3440 PRINT #1;CS :REM CLEAR SCREEN
3450 PRINT #1;"PLEASE ENTER VALUE TO WRITE"
3456 REM THE KEY LAST PRESSED WILL BE PRINTED ON THE SCREEN IF
3457 REM WE DON'T CLEAR THE KCODE LOCATION
3460 POKE KCODE,CLR :INPUT VALUE :REM GET VALUE TO WRITE
3470 PRINT #1;"PLEASE ENTER PLANE NUMBER TO WRITE"
3480 INPUT NUMPLN :REM GET PLANE NUMBER TO WRITE
3490 REM NO CHECK FOR INPUT ERROR SINCE THE ASSEMBLY ROUTINE TRUNCATES
3495 REM CALL ASSEMBLY LANGUAGE ROUTINE
3500 X=USR(SETPLN,VALUE,NUMPLN)
3510 GOTO 1350 :REM GO BACK TO SETUP HANDLER
3520 REM *** END OF "A" COMMAND FOR SETUP ***
3530 REM ******************************************************
3540 REM HANDLER TO WRITE VALUE TO DISPLAY BUFFER
3550 REM ******************************************************
3560 PRINT #1;CS :REM CLEAR SCREEN
3570 PRINT #1;"ENTER THE VALUE TO WRITE TO DISP BUF"
3580 POKE KCODE,CLR :INPUT VALUE :REM GET VALUE TO WRITE
3590 REM NO ERROR CHECKING SINCE ASSEMBLY LANGUAGE ROUTINE TRUNCATES
3595 REM CALL ASSEMBLY LANGUAGE ROUTINE
3600 X=USR(SETBUF,VALUE)
3610 GOTO 1350 :REM GO BACK TO SETUP HANDLER
3615 REM END OF COMMAND "B" OF SETUP INITIAL CONDITIONS HANDLER
3620 REM ******************************************************
3630 REM WRITE A BOX TO THE DISPLAY BUFFER
3640 REM ******************************************************
3650 PRINT #1;CS :REM CLEAR SCREEN
3660 X=USR(SETBUF,0) :REM CLEAR DISPLAY BUFFER
3670 ROW=32 :REM 32 BYTES/ROW ON SIMULATOR MONITOR
3675 REM THE SIMULATOR DISPLAY IS FORMED BY 256 RASTER SCAN LINES
3677 REM WITH 256 PIXELS IN EACH LINE
3678 REM PUT THE BOX IN THE CENTER
3680 INDEX=ROW*120+15 :REM OFFSET FOR LEFT-UPPER CORNER OF BOX
3685 REM PUT THE VALUE 255 (ALL 1'S) IN AN AREA 16 PIXELS WIDE
3687 REM BY 16 PIXELS HIGH IN THE DISPLAY BUFFER
3690 FOR CELL=DISPBUF+INDEX TO DISPBUF+INDEX+16*ROW STEP ROW
3700 POKE CELL,255
3710 POKE CELL+1,255
3720 NEXT CELL
3730 GOTO 1350 :REM BACK TO SETUP HANDLER
3740 REM END OF COMMAND "C" OF SETUP HANDLER
3750 REM******************************************************************************
3760 REM AND MEMORY PLANE WITH DISPLAY BUFFER
3770 REM******************************************************************************
3780 PRINT #1;C$: :REM CLEAR SCREEN
3790 PRINT #1;:"ENTER THE PLANE NUMBER TO AND"
3800 POKE KCODA,CLR :INPUT NUMPLN :REM GET PLANE NUMBER TO AND
3810 REM NO ERROR CHECK BECAUSE TRUNCATION BY ASSEMBLY ROUTINE
3815 REM CALL APPROPRIATE ASSEMBLY LANGUAGE ROUTINE
3820 X=USR(BUFPLN,0,NUMPLN)
3823 REM NOW WRITE THIS VALUE TO THE MEMORY PLANE
3825 X=USR(WRITED,NUMPLN)
3830 REM
3840 GOTO 1350 :REM RETURN TO SETUP HANDLER
3850 REM******************************************************************************
3860 REM OR MEMORY PLANE WITH DISPLAY BUFFER
3870 REM******************************************************************************
3880 PRINT #1;C$: :REM CLEAR SCREEN
3890 PRINT #1;:"ENTER THE PLANE NUMBER TO OR"
3900 POKE KCODA,CLR :INPUT NUMPLN :REM GET # OF PLANE TO OR
3905 REM CALL APPROPRIATE ASSEMBLY LANGUAGE ROUTINE
3910 X=USR(BUFPLN,2,NUMPLN)
3913 REM WRITE THIS VALUE TO THE MEMORY PLANE
3915 X=USR(WRITED,NUMPLN)
3920 REM
3930 GOTO 1350 :REM RETURN TO SETUP HANDLER
3940 REM******************************************************************************
3950 REM EXOR MEMORY PLANE WITH DISPLAY BUFFER
3960 REM******************************************************************************
3970 PRINT #1;C$: :REM CLEAR SCREEN
3980 PRINT #1;:"ENTER THE PLANE NUMBER TO EXOR"
3990 POKE KCODA,CLR :INPUT NUMPLN :REM GET # OF PLANE TO EXOR
4000 REM CALL ASSEMBLY LANGUAGE ROUTINE
4005 X=USR(WRITED,NUMPLN)
4010 GOTO 1350 :REM RETURN TO SETUP HANDLER
4020 REM
4030 REM
4040 REM******************************************************************************
4050 REM MOVE SIMULATOR DISPLAY UP
4060 REM******************************************************************************
4065 REM THE BACKGROUND CONTROL PROCESS (SEE ASSEMBLY LISTING) WILL
4066 REM SEND FUNCTION # 6 COMMAND TO THE SIMULATOR. THE HARDWARE
4067 REM OF THE SIMULATOR WILL MOVE THE DISPLAY UP
4070 POKE NUMFNC,6 :REM FUNCTION 6
4080 POKE PERIOD,0 :REM DO ONCE ONLY
4090 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4100 REM
4110 REM RETURN TO SETUP HANDLER
4120 GOTO 1520
4130 REM ********************************************
4140 REM MOVE DISPLAY SIMULATOR DOWN
4150 REM ********************************************
4160 POKE NUMFNC,4 :REM FUNCTION 4
4170 POKE PERIOD,0 :REM DO ONLY ONCE
4180 POKE CDMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4190 GOTO 1520 :REM RETURN TO SETUP HANDLER
4200 REM ********************************************
4210 REM MOVE SIMULATOR DISPLAY LEFT
4220 REM ********************************************
4230 POKE NUMFNC,5 :REM FUNCTION 5
4240 POKE PERIOD,0 :REM DO ONLY ONCE
4250 POKE CDMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4260 GOTO 1520 :REM RETURN TO SETUP HANDLER
4270 REM ********************************************
4280 REM MOVE SIMULATOR DISPLAY RIGHT
4290 REM ********************************************
4300 POKE NUMFNC,7 :REM FUNCTION 7
4310 POKE PERIOD,0 :REM DO ONLY ONCE
4320 POKE CDMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4330 GOTO 1520 :REM RETURN TO SETUP HANDLER
4340 REM
4350 REM ********************************************
4360 REM RETURN TO TOP LEVEL OF PROGRAM
4370 REM ********************************************
4380 GOTO 470
4390 REM GO TO MAIN LOOP
4400 REM
4410 REM *** END OF SETUP HANDLER ***
4420 REM
4440 REM *** START OF RUN SIMULATOR ***
4450 REM ********************************************
4460 REM PLACES A BOX ON THE SIMULATOR DISPLAY
4470 REM ********************************************
4475 REM THE BACKGROUND CONTROL PROCESS IS USED (SEE ASSEMBLY LISTING)
4477 REM TO PLACE A BOX ON THE SIMULATOR DISPLAY
4480 POKE NUMFNC,0 :REM FUNCTION 0
4490 POKE PERIOD,0 :REM DO ONLY ONCE
4500 POKE CDMV2,1 :REM TURN ON BACKGROUND PROCESS
4510 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4520 REM *** END OF PLACING BOX ***
4530 REM
4540 REM ********************************************
4550 REM RUNS DOWNLOADED TRANSITION RULE
4560 REM ********************************************
4570 REM
4580 POKE NUMFNC,1 :REM FUNCTION 1
4590 POKE PERIOD,1 :REM DO UNTIL USER WANTS TO STOP
4600 POKE CDMV2,1 :REM TURN ON BACKGROUND PROCESS
4610 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4620 REM END OF RUNNING DOWNLOADED TRANSITION RULE
4625 REM
4630 REM ********************************************
4640 REM RUN REVERSIBLE VERSION OF DOWNLOADED TRANSITION RULE
4650 REM *****************************************************
4660 POKE NUMFNC,2 :REM FUNCTION 2
4670 POKE PERIOD,1 :DO UNTIL USER WANTS TO STOP
4680 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4690 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4700 REM
4710 REM
4720 REM ********************************************************
4730 REM RUN "LIFE"
4740 REM ********************************************************
4750 POKE NUMFNC,3 :REM FUNCTION 3
4760 POKE PERIOD,1 :REM RUN UNTIL USER WANTS TO STOP
4770 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4780 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4790 REM
4800 REM ********************************************************
4810 REM MOVES SIMULATOR DISPLAY UP
4820 REM ********************************************************
4830 POKE NUMFNC,6 :REM FUNCTION 6 MOVES DISPLAY UP
4840 POKE PERIOD,0 :REM DO ONLY ONCE
4850 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4860 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4870 REM
4880 REM ********************************************************
4890 REM MOVE SIMULATOR DISPLAY DOWN
4900 REM ********************************************************
4920 POKE NUMFNC,4 :REM FUNCTION 4 MOVES DISPLAY DOWN
4930 POKE PERIOD,0 :REM DO ONCE ONLY
4940 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
4945 REM REFER TO ASSEMBLY LANGUAGE ROUTINE FOR CLARIFICATION
4950 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
4960 REM
4970 REM
4980 REM ********************************************************
4990 REM MOVE SIMULATOR DISPLAY LEFT
5000 REM ********************************************************
5010 POKE NUMFNC,5 :REM FUNCTION 5 MOVES DISPLAY TO THE LEFT
5020 POKE PERIOD,0 :REM DO ONLY ONCE
5030 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
5040 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
5050 REM
5060 REM
5070 REM ********************************************************
5080 REM MOVE SIMULATOR DISPLAY RIGHT
5090 REM ********************************************************
5100 POKE NUMFNC,7 :REM FUNCTION 7 MOVES DISPLAY RIGHT
5110 POKE PERIOD,0 :REM DO ONLY ONCE
5120 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
5130 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
5140 REM
5150 REM
5160 REM ********************************************************
5170 REM CHANGE SPEED OF SIMULATOR
5180 REM ********************************************************
5190 POSITION 2,19 :REM PLACE CURSOR AT LOWER-LEFT
5200 PRINT #1;"ENTER THE PERIOD IN 1/60 SEC BETWEEN"
5210 PRINT #1;"GENERATIONS. DEFAULT PERIOD IS 1/60"
5213 REM WRITE PROMPTS TO USER
5215 REM NEED TO CLEAR KEY CODE TO SUPPRESS PRINTING OF THE KEY
5217 REM THAT WAS Pressed TO GET TO THIS ROUTINE
5220 POKE KORDER,CLR :INPUT TIME, :REM GET NEW PERIOD
5230 POKE PERIOD,TIME :REM THIS CHANGES THE PERIOD (SPEED) OF
5235 REM THE SIMULATOR. REFER TO THE ASSEMBLY LISTING
5240 POSITION 2,19 :REM CLEAN UP SCREEN BY OVERWRITING PREVIOUS
5245 REM PRINT STATEMENTS
5250 PRINT #1;"PLEASE ENTER THE COMMAND YOU WISH:"
5260 PRINT #1;"
5270 PRINT #1;"
5280 GOTO 2070 :REM GO BACK TO RUN SIMULATOR HANDLER
5290 REM
5300 REM *****************
5310 REM STOP SIMULATOR
5320 REM *****************
5330 POKE CDMV2,0 :REM STOP SIMULATOR BY TURNING OFF BACKGROUND
5340 POKE PERIOD,0 :REM CONTROL PROCESS
5350 GOTO 2070 :REM GO BACK TO TOP LEVEL OF RUN SIMULATOR
5360 REM HANDLER
5370 REM
5380 REM *****************
5390 REM JUMP TO TOP LEVEL OF PROGRAM
5400 REM *****************
5410 GOTO 470 :REM RETURN TO TOP LEVEL
5420 REM
5430 REM
5440 REM
5450 REM *****************
5460 REM TRY TO REVERSE SIMULATOR
5470 REM *****************
5471 REM EACH PIXEL OF THE SIMULATOR DISPLAY IS FORMED BY A BIT-MAP
5472 REM FROM 2 MEMORY PLANES, HENCE EACH PIXEL HAS 4 POSSIBLE GREY
5473 REM GREY LEVELS, 1 MEMORY PLANE HOLDS THE VALUE FOR THE PRESENT
5474 REM STATE AND THE OTHER THE VALUES FOR THE PREVIOUS STATE.
5475 REM IF WE FORM THE NEXT STATE OF THE CENTER CELL BY APPLYING
5476 REM THE CURRENT TRANSITION RULE EXOR'D WITH THE PREVIOUS STATE
5477 REM WE FORM A "REVERSIBLE" RULE, WE CAN THEN MAKE THE SIMULATOR
5478 REM GO BACKWARDS BY SWAPPING THE MEMORY PLANES FOR THE PRESENT
5479 REM AND PREVIOUS STATES AND RUNNING THE REVERSIBLE RULE
5480 X=USR(RDSDP,0) :REM READ PLANE 0,CURRENT STATE INTO DISP BUFFER
5490 OPEN #2,8,0,"D:BUF"
5500 X=USR(SAVBUF,2) :REM OPEN A FILE AND STORE PLANE 0 ON DISK
5510 REM
5520 X=USR(RDSDP,1) :REM READ PLANE 1,PAST STATE INTO DISPLAY BUFFER
5530 REM
5540 X=USR(WRITED,0) :REM WRITE PLANE 1 TO PLANE ON
5550 REM
5560 REM CLOSE FILE
5570 CLOSE #2 :REM NOW SEND PLANE 0 TO PLANE 1
5580 OPEN #2,4,0,"D:BUF"
5590 REM OPEN FILE WHICH HAS PLANE 0 STORED IN IT
5600 X=USR(LODBF,2) :REM LOAD DISPLAY BUFFER WITH THIS FILE
5810 X=USR(WRITE,1) :REM WRITE PLANET O TO PLANET 1
5820 REM
5830 CLOSE #2 :REM CLOSE FILE
5840 GOTO 2070 :GOTO TOP LEVEL OF RUN SIMULATOR ROUTINE
5850 REM USER CAN NOW TRY TO REVERSE SIMULATOR BY ENTERING
5855 REM THE COMMAND FOR RUNNING THE REVERSIBLE TRANSITION RULE
5860 REM
5870 REM ******************************************************
5875 REM RUNS TRANSITION RULES IN SINGLE-STEP MODE
5880 REM ******************************************************
5890 POKE CDTMV2.0
5900 POKE PERIOD,0
5910 REM TURN OFF SIMULATOR IF NOT ALREADY OFF
5920 POSITION 2,19 :REM GIVE USER PROMPTS
5930 PRINT #1;"ENTER 'P','Q' OR 'R' TO SINGLE STEP"
5940 PRINT #1;"THROUGH COMMANDS B,C,D RESPECTIVELY"
5950 PRINT #1;"ANY OTHER CHAR TO RUN OTHER SIM FUNCS"
5960 POKE KCODE,CLR :REM CLEAR KEY CODE
5970 CHAR=PEEK(KCODE) :REM WAIT FOR INPUT
5980 IF CHAR=CLR THEN GOTO 5870
5985 REM JUMP TABLE
5990 IF CHAR=P THEN GOTO 5990
6000 IF CHAR=Q THEN GOTO 6150
6010 IF CHAR=R THEN GOTO 6310
6015 REM ELSE CLEAN UP SCREEN AND RETURN TO RUN SIMULATOR HANDLER
6020 POSITION 2,19
6030 PRINT #1;"
6040 PRINT #1;"
6050 PRINT #1;"
6060 REM
6070 REM "P" COMMAND GETS VECTORED HERE
6080 POKE NUMFNC,1 :REM FUNCTION 1 IS > RUN TRANSITION TABLE
6090 POKE PERIOD,0 :REM DO 1 STEP ONLY
6100 POKE CDTMV2,1 :REM TURN ON BACKGROUND CONTROL PROCESS
6110 POSITION 2,19 :REM PROMPT USER
6120 PRINT #1;"ENTER 'P' TO CONTINUE SINGLE-STEPING"
6130 PRINT #1;"ANY OTHER CHAR TO RUN OTHER SIM FUNCS"
6140 PRINT #1;"
6150 POKE KCODE,CLR
6160 CHAR=PEEK(KCODE)
6170 IF CHAR=CLR THEN GOTO 6070
6180 REM IF CHAR=P THEN CONTINUE SINGLE-STEPING
6190 IF CHAR=P THEN GOTO 5990
6200 POSITION 2,19
6210 PRINT #1;"
6220 PRINT #1;"
6230 PRINT #1;"
6240 REM ELSE CLEAN UP SCREEN AND RETURN TO RUN SIMULATOR HANDLER
6250 GOTO 2060
6260 REM "Q" COMMAND OF SINGLE-STEP THROUGH RULE GETS VECTORED HERE
6270 POKE NUMFNC,2 :REM FUNCTION 2 IS RUN REVERSIBLE TRANS RULE
6280 POKE PERIOD,0 :REM DO 1 STEP ONLY
6290 POKE CDTMV2,1 :REM TURN ON BACKGROUND PROCESS
6300 POSITION 2,19 :REM PROMPT USER
6310 PRINT #1;"ENTER 'Q' TO CONTINUE SINGLE-STEPING"
6200 PRINT #1;"ANY OTHER CHAR TO RUN OTHER SIM FUNCS"
6210 PRINT #1;"
6215 REM CONTINUE SINGLE STEPPING AS LONG AS "Q" IS ENTERED
6220 POKE KCODE,CLR
6230 CHAR=PEEK(KCODE)
6240 IF CHAR=CLR THEN GOTO 6230
6250 IF CHAR=Q THEN GOTO 6150
6255 REM ELSE CLEAN UP SCREEN AND
6260 POSITION 2,19
6270 PRINT #1;"
6280 PRINT #1;"
6290 REM THEN RETURN TO TOP LEVEL OF RUN SIMULATOR HANDLER
6300 GOTO 2060
6305 REM COMMAND "Q" OF SINGLE-STEP HANDLER GETS VECTORED HERE
6310 POKE NUMFNC,3 :REM FUNCTION 3 IS RUN "LIFE"
6320 POKE PERIOD,0 :REM DO 1 STEP ONLY
6330 POKE CDTMV,2,1 :REM TURN ON BACKGROUND PROCESS
6340 POSITION 2,19 :REM PROMPT USER
6350 PRINT #1;"ENTER 'R' TO CONTINUE SINGLE-STEPPING"
6360 PRINT #1;"ANY OTHER CHAR TO RUN OTHER SIM FUNCS"
6370 PRINT #1;"
6375 REM CONTINUE SINGLE-STEPPING THROUGH "LIFE" IF "R" IS ENTERED
6380 POKE KCODE,CLR
6390 CHAR=PEEK(KCODE)
6400 IF CHAR=CLR THEN GOTO 6390
6410 IF CHAR=R THEN GOTO 6310
6415 REM ELSE CLEAN UP SCREEN AND
6420 POSITION 2,19
6430 PRINT #1;"
6440 PRINT #1;"
6450 REM THEN RETURN TO TOP LEVEL OF RUN SIMULATOR HANDLER
6460 GOTO 2060
6470 REM
6480 REM
6490 REM END OF RUN SIMULATOR ROUTINE
6500 REM *****************************************************
6510 REM HANDLER FOR SAVING TRANSITION RULES
6520 REM *****************************************************
6530 PRINT #1;C$:REM PROMPT USER
6540 PRINT #1;"ENTER THE NAME YOU WISH TO SAVE THE"
6550 PRINT #1;"TRANSITION RULE UNDER:"
6560 POKE KCODE,CLR :INPUT T$:REM GET NAME TO SAVE RULE UNDER
6570 D$="D:" :REM PUT THE FILE NAME INTO THE PROPER FORMAT BY
6580 D$(3)=T$:REM CONCATENATING THE FILE NAME TO "D:"
6590 REM NOW SAVE THE DEFINITION OF THE TRANSITION RULE TO DISK
6600 LIST D$,30000,30010
6610 REM
6620 REM GO BACK TO TOP LEVEL OF SAVE/RETRIEVE TRANSITION RULE HANDLER
6630 GOTO 2320
6640 REM
6650 REM *****************************************************
6660 REM SAVE TRANSITION TABLE
6670 REM *****************************************************
6680 PRINT #1;C$:REM CLEAR SCREEN AND PROMPT USER
6690 PRINT #1;"ENTER THE NAME YOU WISH TO SAVE THE"
6700 PRINT #1;"TRANSITION TABLE UNDER:"
6710 POKE KCODE, CLR : INPUT T$ : REM GET NAME TO SAVE TABLE UNDER
6720 D$="D:" : REM FORMAT INTO PROPER FORM FOR WRITING TO
6730 D$(3)=T$ : REM DISK BY CONCATENATING WITH HEADER "D:"
6740 OPEN #3,8,0,D$ : REM OPEN A CHANNEL TO DISK DRIVE
6750 PRINT #3;SAVEX$ : REM AND STORE SAVEX$ WHICH IS THE TRANSITION
6760 CLOSE #3 : REM TABLE IN STRING FORM ON DISK.
6770 GOTO 2320 : REM CLOSE CHANNEL AND GO BACK TO TOP LEVEL
6780 REM OF THE SAVE/RETRIEVE TRANSITION RULE HANDLER
6790 REM
6800 REM
6810 REM ***********************************************
6820 REM RETRIEVE TRANSITION RULE
6830 REM ***********************************************
6840 PRINT #1;C$ : REM CLEAR SCREEN AND PROMPT USER
6850 PRINT #1;"ENTER THE NAME THAT THE RULE IS SAVED"
6860 PRINT #1;"UNDER:"
6870 POKE KCODE, CLR : INPUT T$ : REM GET NAME OF RULE TO
6880 D$="D:" : REM AND PUT IN REQUIRED FORMAT FOR DISK
6890 D$(3)=T$ : REM BY CONCATENATING WITH "D:"
6900 GOSUB 9200 : REM SETUP FOR READING IN THIS RULE
6910 ENTER D$ : REM WHEN SETUP COMPLETED ENTER THE FILE UNDER
6920 REM WHICH THE TRANSITION RULE WAS SAVED UNDER
6930 GOSUB 3170 : REM NOW DOWNLOAD THIS TRANSITION RULE TO THE
6940 GOTO 2320 : REM SIMULATOR AND RETURN TO TOP LEVEL OF THE
6950 REM SAVE/RETRIEVE TRANSITION RULE HANDLER
6960 REM
6970 REM
6980 REM ***********************************************
6990 REM REtrieve Transition table
7000 REM ***********************************************
7010 PRINT #1;C$ : REM CLEAR SCREEN AND PROMPT USER
7020 PRINT #1;"ENTER THE NAME THAT THE TRANSITION"
7030 PRINT #1;"UNDER:"
7040 POKE KCODE, CLR : INPUT T$ : REM GET NAME OF FILE WHICH HAS
7050 D$="D:" : REM THE TRANSITION TABLE AND PUT IN THE PROPER
7060 D$(3)=T$ : REM FORMAT BY CONCATENATING WITH "D:"
7070 TRAP 9400 : REM SET TRAP IN CASE USER ASKS FOR A NON
7080 OPEN #3,4,0,D$ : REM NEW OPEN FILE
7090 INPUT #3;SAVEX$ : REM READ IN TRANSITION TABLE
7100 CLOSE #3 : REM CLOSE CHANNEL TO DISK DRIVE
7110 REM SETUP DATA TRANSFER BETWEEN ATARI AND SIMULATOR
7120 X=USR(SETDAT,32,255)
7130 X=USR(STPDAT) : REM SAME KLUGE AS MENTIONED BEFORE
7140 X=USR(SETDAT,32,255)
7150 REM 32=LOAD TRANSITION RULE,255=DATA TRANSFER IS OUTPUT
7160 OPEN #3,4,0,D$ : REM FOR EVERY POSSIBLE STATE OF N,S,W,E,C DOWNLOAD TABLE
7170 FOR INDEX=1 TO 32 STEP 1
7180 TS$=SAVES(INDEX,INDEX) : REM GET THE CURRENTLY INDEXED ELEMENT
7190 X=ASC(TS$) : REM OF THE STRING AND CONVERT INTO ASCII
7200 POKE PORTB,X : REM X=0 OR 1 FROM THE WAY SAVEX$ WAS FORMED
7210 POKE PORTB,0 : REM SYNC SIMULATOR
7220 NEXT INDEX
7230 X=USR(STPDAT) : REM DISABLE DATA TRANSFER WHEN DONE
7240 REM"GO BACK TO TOP LEVEL OF SAVE/RETRIEVE
7250 GOTO 2320 : REM TRANSITION RULE HANDLER
7190 REM ********************
7200 REM SAVES DISPLAY BUFFER ON DISK
7210 REM ********************
7220 PRINT #1;C$ :REM CLEAR SCREEN AND PROMPT USER
7230 PRINT #1;"ENTER THE NAME YOU WISH TO SAVE THE"
7240 PRINT #1;"DISPLAY BUFFER UNDER"
7250 POKE KCODE,CLR :INPUT T$ :REM GET NAME TO SAVE DISPLAY
7260 D$="" :REM BUFFER UNDER AND FORMAT IT FOR DISK
7270 D$(3)=T$: :REM ACCESS BY CONCATENATING WITH "D:"
7280 OPEN #3,8,0,D$ :OPEN CHANNEL TO DISK DRIVE
7290 X=USR(SAVBUF,3) :REM SAVE BUFFER ON DISK
7300 CLOSE #3 :REM CLOSE CHANNEL
7310 GOTO 2580 :REM GO BACK TO TOP LEVEL OF SAVE/RETRIEVE
7320 REM DISPLAY BUFFER HANDLER
7330 REM
7340 REM
7350 REM ********************
7360 REM LOAD DISPLAY BUFFER FROM DISK
7370 REM ********************
7380 PRINT #1;C$ :REM CLEAR SCREEN AND PROMPT USER
7390 PRINT #1;"ENTER THE NAME OF THE FILE YOU WISH"
7400 PRINT #1;"TO LOAD THE DISPLAY BUFFER WITH:"
7405 REM GET NAME OF FILE TO LOAD DISPLAY BUFFER WITH
7410 POKE KCODE,CLR :INPUT T$
7420 D$="D:" :REM FORMAT FOR DISK ACCESS BY CONCATENATION
7430 D$(3)=T$: :REM WITH "D:"
7435 TRAP 9400 :REM SET TRAP FOR POSSIBILITY OF NONEXISTANT
7440 OPEN #3,4,0,D$ :REM FILE THEN OPEN CHANNEL TO DISK DRIVE
7450 X=USR(LODBF,3) :REM LOAD DISPLAY BUFFER FROM SPECIFIED FILE
7460 CLOSE #3 :REM CLOSE CHANNEL
7470 GOTO 2580 :REM RETURN TO TOP LEVEL OF SAVE/RETRIEVE
7480 REM DISPLAY BUFFER HANDLER
7490 REM
7500 REM ********************
7510 REM WRITE DISPLAY BUFFER TO MEMORY PLANE
7520 REM ********************
7530 PRINT #1;C$ :REM CLEAR SCREEN AND PROMPT USER
7540 PRINT #1;"ENTER THE PLANE NUMBER TO WRITE TO:"
7545 REM GET NUMBER OF PLANE TO WRITE TO
7550 POKE KCODE,CLR :INPUT NUMPLN
7560 X=USR(WRITED,NUMPLN) :REM WRITE DISPLAY BUFFER TO MEMORY
7570 GOTO 2890 :REM PLANE SPECIFIED AND RETURN TO TOP LEVEL
7580 REM OF WRITE/READ MEMORY PLANE HANDLER
7590 REM
7600 REM ********************
7610 REM READ MEMORY PLANE
7620 REM ********************
7630 PRINT #1;C$ :REM CLEAR SCREEN AND PROMPT USER
7640 PRINT #1;"ENTER THE PLANE NUMBER TO READ FROM:"
7645 REM GET NUMBER OF PLANE TO READ FROM
7650 POKE KCODE,CLR :INPUT NUMPLN
7660 X=USR(RDDSP,NUMPLN) :REM READ THE MEMORY PLANE TO THE DISPLAY
7670 GOTO 2890 :REM BUFFER AND RETURN TO TOP LEVEL OF THE
7680 REM READ/WRITE MEMORY PLANE FUNCTION HANDLER
7690 REM
7700 REM ********************
REM NEW ROUTINE CAN BE PLACED HERE ; NOTHING AT MOMENT
REM
GOTO 470
REM FOR NOW GO BACK TO TOP LEVEL OF PROGRAM
REM
REM START OF HELP ROUTINE
REM
PRINT #1,CS  :REM GET PROMPTS FROM THE FILE "D:HELP.PMT"
FNAME$="D:HELP.PMT"
GOSUB 9000  :REM AND PRINT IT ON THE SCREEN
PRINT #1;"ENTER ANY KEY TO RETURN TO TOP LEVEL"
PKE KCODE,CLR  :REM CLEAR KEY CODE
CHAR=PEEK(KCODE)  :REM THEN RETURN TO TOP LEVEL OF PROGRAM
IF CHAR=CLR THEN GOTO 8240
GOTO 470  :REM WHEN USER HAS FINISHED READING
REM
EXIT FROM PROGRAM
REM
PRINT #1,CS  :REM CLEAR SCREEN AND PROMPT USER
PRINT #1;"ENTER 'CONT' FOLLOWED BY A <CR> TO"
PRINT #1;"RESTART PROGRAM"
PKE KCODE,CLR  :REM CLEAR KEY CODE
STOP  :REM EXIT PROGRAM
GOTO 470
REM IF THE PROGRAM IS RESTARTED IT WILL START AT TOP LEVEL
REM
SUBROUTINE TO READ IN USER PROMPTS FROM DISK
REM THE FILE NAME TO READ TEXT FROM IS PASSED TO ROUTINE BY
REM FNAME$. OPEN CHANNEL TO DISK DRIVE
OPEN #4,4,0,FNAME$
TRAP 9100  :REM THERE IS NO EOF MARKER SO SET TRAP TO
DETECT AN END OF FILE CONDITION
INPUT #4,TXT$  :REM READ IN TEXT; USEFUL PART IS ONLY FROM
REM THE 9TH ELEMENT TO THE 45TH ELEMENT. SO PRINT THIS PART
REM ONLY TO THE SCREEN
PRINT #1;TXT$(9,45)
GOTO 9020  :REM KEEP ON READING IN TEXT UNTIL THERE IS AN
REM END OF FILE ERROR
REM IN WHICH CASE WE GO THE TRAP ROUTINE AT 9100
REM
TRAP OCCURS WHEN THERE IS NO MORE DATA TO READ
TRAP 40000  :REM CLEAR TRAP
CLOSE #4  :REM CLOSE CHANNEL TO DISK
RETURN  :REM AND RETURN
REM
REM THIS SUBROUTINE IS CALLED BEFORE AN "ENTER FILE" COMMAND
REM WHEN AN "ENTER FILE" COMMAND IS ENCOUNTERED THAT FILE
REM IS LOADED INTO RAM AND AN EXIT IS MADE FROM THE PROGRAM
REM FURTHERMORE,BASIC WILL PRINT "READY".
REM BY PLACING THE CURSOR IN AN ILLEGAL RANGE
REM WE WILL FORCE THE PROGRAM TO CONTINUE BY TRAPPING THE ERROR
REM THAT WILL OCCUR WHEN BASIC HAS FINISHED ENTERING THE FILE
REM AND PRINTS "READY"
REM PLACE CURSOR BACK IN ILLEGAL RANGE
POS 2,100

SET TRAP
POS 330

REM END OF SUBROUTINE
POS 315

REM ERROR TRAPPED TO NEXT LINE
POS 320

REM PLACE CURSOR BACK IN A VALID RANGE
POS 330

REM CLEAR TRAP
POS 340

REM RETURN TO ONLY PLACE THIS TYPE OF ERROR
POS 350

REM COULD OCCUR AT
POS 360

REM INCORRECT FILE NAME ERROR TRAPPED HERE
POS 380

REM GIVE USER ERROR MESSAGE
POS 390

PRINT #1;"ERROR";PEEK(ERRSV);"ON LINE";PEEK(STOPLN)+256*PEEK(STOPLN+1)
POS 400

PRINT #1;"PROBABLY AN INCORRECT FILE NAME"
POS 410

PRINT #1;"ENTER ANY CHAR TO RETURN TO TOP LEVEL"
POS 420

CHAR=PEEK(KCODE)
POS 430

IF CHAR=CLR THEN GOTO 9430
POS 440

GOTO 470:REM RETURN TO TOP LEVEL
POS 440

REM LINES 30000 TO 30020 FORM A SUBROUTINE TO CALCULATE
POS 30000

REM THE NEXT STATE OF THE CENTER CELL GIVEN THE CURRENT
POS 30000

REM HERE IS AN EXAMPLE
POS 30000

REM BANKS RULE
POS 30000

TEMP=N+S+W+E:X=C
POS 30000

IF TEMP=4 OR TEMP=3 THEN X=1
POS 30000

TEST=(N AND E)+(E AND S)+(S AND W)+(W AND N)
POS 30000

IF TEMP=2 AND TEST THEN X=0
POS 30000

RETURN
The following is the contents of the file "D:TOPLVL.PMT". It informs the user of what commands are available at the top level of the program. The line numbers the word REM and the quotation marks have no particular significance; they are stripped off by the Basic program.

10 REM "THIS PROGRAM CONTROLS THE CELLULAR"
11 REM "AUTOMATA SIMULATOR"
12 REM "WHEN A COMMAND IS INVOKED DOCUMENTATION EXPLAINING THAT COMMAND WILL"
13 REM "APPEAR"
14 REM "THE FOLLOWING COMMANDS ARE AVAILABLE:"
15 REM "A: DEFINE A TRANSITION RULE"
16 REM "B: SETUP INITIAL CONDITION FOR SIM"
17 REM "C: RUN THE SIMULATOR"
18 REM "D: SAVE/RETRIEVE A TRANSITION RULE"
19 REM "E: SAVE/RETRIEVE THE DISPLAY BUFFER"
20 REM "F: READ OR WRITE TO A MEMORY PLANE"
21 REM "G: NOTHING YET"
22 REM "H: HELP"
23 REM "I: EXIT THE PROGRAM"
24 REM ""
25 REM "PLEASE ENTER THE COMMAND YOU WISH:"

The following are the contents of the file "D:HELP.PMT".
It informs the user in general terms how to use the program.

10 REM "THE USER SHOULD SETUP THE INITIAL
11 REM "CONDITIONS OF THE SIMULATOR DISPLAY
12 REM "USING COMMAND B AT TOP LEVEL. THEN
13 REM "THE TRANSITION RULE SHOULD BE DOWN-
14 REM "LOADED TO THE SIMULATOR. AT FIRST
15 REM "COMMAND A AT THE TOP LEVEL OF THE
16 REM "PROGRAM WOULD BE THE BEST CHOICE
17 REM "TO DO THIS, AT LEAST UNTIL A LIBRARY
18 REM "OF TRANSITION RULES IS CONSTRUCTED
19 REM "THEN USE COMMAND C TO RUN THE TRA-
20 REM "SION RULE ON THE INITIAL CONDI-
21 REM "TION SETUP. THE OTHER COMMANDS ARE
22 REM "VARIATIONS ON THIS THEME, GENERALLY
23 REM "SPEAKING."
The following are the contents of the file "D:SETUPSIM.PMT"
It informs the user of the various commands available to set up the initial state of the simulator display. Only the text within the quotation marks is printed to the screen. This is true of all the other files also.

10 REM "THE FOLLOWING COMMANDS ARE AVAILABLE"
11 REM "TO SETUP THE INITIAL CONDITIONS OF"
12 REM "THE SIMULATOR"
13 REM ""
14 REM "A : WRITE A VALUE TO THE MEMORY PLANE"
15 REM "B : WRITE A VALUE TO THE DISPLAY BUF"
16 REM "C : WRITE A BOX TO THE DISPLAY"
17 REM "D : AND MEMORY PLANE WITH DISP BUFFER"
18 REM "E : OR MEMORY PLANE WITH DISP BUFFER"
19 REM "F : EXOR MEMORY PLANE WITH DISP BUFR"
20 REM "G : MOVE DISPLAY UP"
21 REM "H : MOVE DISPLAY DOWN"
22 REM "I : MOVE DISPLAY LEFT"
23 REM "J : MOVE DISPLAY RIGHT"
24 REM "K : RETURN TO TOP LEVEL OF PROGRAM"
25 REM "ENTER THE COMMAND YOU WISH"
The following are the contents of the file "D:TRNSTN.PMT". It informs the user how to define a transition rule for the simulator. Only the text within the quotation marks are printed to the screen.

10 REM "THE TRANSITION RULE TO DETERMINE THE "
11 REM "NEXT STATE OF THE CENTER CELL (C)"
12 REM "FROM THE CURRENT STATE OF ITS"
13 REM "4 NEIGHBORS (N,S,W,E) AND THE CURRENT"
14 REM "STATE OF THE CENTER CELL CAN BE"
15 REM "DEFINED IN THE FOLLOWING WAY:"
16 REM ""
17 REM "THE USER MUST USE THE SCREEN EDITOR"
18 REM "TO DEFINE A VARIABLE, X, IN TERMS OF"
19 REM "VARIABLES N,S,W,E AND C IN LINES"
20 REM "30000 TO 30010, WHERE X SHOULD"
21 REM "DETERMINE THE NEXT STATE OF THE"
22 REM "CENTER CELL"
The following are the contents of the file "D:RUNSIM1.PMT". It gives the user a reminder to setup the initial conditions of the simulator and to download a transition rule before attempting to run the simulator.

10 REM "BEFORE RUNNING THE SIMULATOR IT"
11 REM "MIGHT BE ADVISABLE TO SETUP INITIAL"
12 REM "CONDITIONS FOR THE SIMULATOR BY"
13 REM "USING THE 'B' COMMAND AT THE TOP"
14 REM "LEVEL OF THE PROGRAM"
15 REM "ALSO, THE TRANSITION RULE TO BE USED"
16 REM "SHOULD HAVE BEEN DOWNLOADED"
17 REM ""
18 REM "ENTER 'Q' TO RETURN TO TOP LEVEL"
19 REM "ANY OTHER CHARACTER TO CONTINUE"
The following are the contents of the file "D:RUNSIM2.PMT". It informs the user of what commands are available to run the simulator.

10 REM "THE FOLLOWING COMMANDS ARE AVAILABLE"
11 REM "TO CONTROL THE SIMULATOR:"
12 REM ""
13 REM "A : PLACE A BOX IN THE SIMULATOR DISP"
14 REM "B : RUNS DOWNLOADED TRANSITION RULE"
15 REM "C : RUNS A REVERSIBLE VERSION OF RULE"
16 REM "D : RUNS 'LIFE'"
17 REM "E : MOVES DISP UP"
18 REM "F : MOVES DISP DOWN"
19 REM "G : MOVES DISP LEFT"
20 REM "H : MOVES DISP RIGHT"
21 REM "I : CHANGE SPEED OF SIMULATOR"
22 REM "J : STOP SIMULATOR"
23 REM "K : RETURN TO TOP LEVEL OF PROGRAM"
24 REM "L : TRY TO REVERSE SIMULATOR"
25 REM "M : DISPLAY BLOWUP ON TERMINAL SCREEN"
26 REM "N : RUNS CMDFS B,C,D 1 STEP AT A TIME"
27 REM "O : RETURN TO TOP LEVEL"
28 REM "PLEASE ENTER YOUR COMMAND"
The following is the contents of the file "D:SAVRTVTR.PMT". It informs the user of what commands are available to save or retrieve transition rules for the simulator.

10 REM "THE FOLLOWING COMMANDS ARE AVAILABLE"
11 REM "TO SAVE OR RETRIEVE A TRANSITION RULE"
12 REM ""
13 REM "A: SAVES DEF'N OF TRANSITION RULE"
14 REM "B: SAVES THE TRANSITION TABLE ONLY"
15 REM "C: RETRIEVES RULE AND DOWNLOADS IT"
16 REM "D: RETRIEVES TABLE AND DOWNLOADS IT"
17 REM "E: RETURN TO TOP LEVEL"
The following are the contents of the file "D:SAVRTVMP.PMT".
It informs the user what commands are available to read or
write to the memory planes of the simulator.

10 REM "THE FOLLOWING COMMANDS ARE AVAILABLE "
11 REM "TO READ/WRITE TO THE MEMORY PLANES: "
12 REM "
13 REM "A : WRITE THE DISP BUFF TO MEM PLANE "
14 REM "B : READ MEM PLANE TO THE DISP BUFFER"
15 REM "C : RETURN TO TOP LEVEL"
16 REM "
17 REM "COMMAND 'A' LOADS THE DISPLAY BUFFER"
18 REM "FROM THE MEMORY PLANE USED BY THE"
19 REM "SIMULATOR TO GENERATE ITS DISPLAY"
20 REM "THE INVERSE OPERATION IS CARRIED OUT"
21 REM "BY COMMAND 'A'"
The following are the contents of the file "D:SAVRTVDB.PMT". It informs the user of what commands are available to save or retrieve the display buffer from the disk drive.

10 REM "THE FOLLOWING COMMANDS ARE AVAILABLE"
11 REM "TO SAVE/RETRIEVE THE DISPLAY BUFFER"
12 REM "FROM THE DISK DRIVE:
13 REM ""
14 REM "A : SAVES THE DISPLAY BUFFER ON DISK"
15 REM "B : LOADS THE DISP BUFFER FROM DISK"
16 REM "C : RETURN TO TOP LEVEL OF PROGRAM"
17 REM ""
18 REM "TO SAVE THE CURRENT DISPLAY ON THE"
19 REM "SIMULATOR, THE MEMORY PLANE BEING USED"
20 REM "TO GENERATE THAT DISPLAY MUST BE"
21 REM "LOADED INTO THE DISP BUFFER FIRST"
22 REM "BY USING COMMAND 'F' AT TOP LEVEL"
23 REM "BEFORE USING THE ABOVE 'A' COMMAND"
24 REM "SIMILAR STEPS SHOULD BE TAKEN TO"
25 REM "SEND A STORED DISPLAY TO THE SIM"